Table of Contents

Tensorflow Tutorials using Jupyter Notebook

> From: sjchoi86/Tensorflow-101

TensorFlow tutorials written in Python (of course) with Jupyter Notebook. Tried to explain as kindly as possible, as these tutorials are intended for TensorFlow beginners. Hope these tutorials to be a useful recipe book for your deep learning projects. Enjoy coding! :)

Contents

1. Basics of TensorFlow / MNIST / Numpy / Image Processing / Generating Custom Dataset
2. Machine Learing Basics with TensorFlow: Linear Regression / Logistic Regression with MNIST / Logistic Regression with Custom Dataset
3. Multi-Layer Perceptron (MLP): Simple MNIST / Deeper MNIST / Xavier Init MNIST / Custom Dataset
4. Convolutional Neural Network (CNN): Simple MNIST / Deeper MNIST / Simple Custom Dataset / Basic Custom Dataset
5. Using Pre-trained Model (VGG): Simple Usage / CNN Fine-tuning on Custom Dataset
6. Recurrent Neural Network (RNN): Simple MNIST / Char-RNN Train / Char-RNN Sample / Hangul-RNN Train / Hangul-RNN Sample
7. Word Embedding (Word2Vec): Simple Version / Complex Version
8. Auto-Encoder Model: Simple Auto-Encoder / Denoising Auto-Encoder / Convolutional Auto-Encoder (deconvolution)
9. Class Activation Map (CAM): Global Average Pooling on MNIST
10. TensorBoard Usage: Linear Regression / MLP / CNN
11. Semantic segmentation
12. Super resolution (in progress)
13. Web crawler
14. Gaussian process regression
15. Neural Style
16. Face detection with OpenCV

Requirements

- TensorFlow
- Numpy
- SciPy
- Pillow
- BeautifulSoup
- Pretrained VGG: inside 'data/' folder

Note

Most of the codes are simple refactorings of Aymeric Damien's Tutorial or Nathan Lintz's Tutorial. There could be missing credits. Please let me know.

Collected and Modifyed by Sungjoon

info

BASIC TENSORFLOW

LOAD PACKAGES

```
import numpy as np
import tensorflow as tf
print ("PACKAGES LOADED")
```

```
PACKAGES LOADED
```

SESSION

```
sess = tf.Session()
print ("OPEN SESSION")
```

```
OPEN SESSION
```

TF CONSTANT

```
def print_tf(x):
    print("TYPE IS\n %s" % (type(x)))
    print("VALUE IS\n %s" % (x))
hello = tf.constant("HELLO. IT'S ME. ")
print_tf(hello)
```

```
TYPE IS
 <class 'tensorflow.python.framework.ops.Tensor'>
VALUE IS
 Tensor("Const:0", shape=(), dtype=string)
```

TO MAKE THINKS HAPPEN

```
hello_out = sess.run(hello)
print_tf(hello_out)
```

```
TYPE IS
 <type 'str'>
VALUE IS
 HELLO. IT'S ME.
```

OTHER TYPES OF CONSTANTS

```
a = tf.constant(1.5)
b = tf.constant(2.5)
print_tf(a)
print_tf(b)
```

```
TYPE IS
 <class 'tensorflow.python.framework.ops.Tensor'>
VALUE IS
 Tensor("Const_1:0", shape=(), dtype=float32)
TYPE IS
 <class 'tensorflow.python.framework.ops.Tensor'>
VALUE IS
 Tensor("Const_2:0", shape=(), dtype=float32)
```

```
a_out = sess.run(a)
b_out = sess.run(b)
print_tf(a_out)
print_tf(b_out)
```

```
TYPE IS
 <type 'numpy.float32'>
VALUE IS
 1.5
TYPE IS
 <type 'numpy.float32'>
VALUE IS
 2.5
```

OPERATORS

```
a_plus_b = tf.add(a, b)
print_tf(a_plus_b)
```

```
TYPE IS
 <class 'tensorflow.python.framework.ops.Tensor'>
VALUE IS
 Tensor("Add:0", shape=(), dtype=float32)
```

```
a_plus_b_out = sess.run(a_plus_b)
print_tf(a_plus_b_out)
```

```
TYPE IS
 <type 'numpy.float32'>
VALUE IS
 4.0
```

```
a_mul_b = tf.mul(a, b)
a_mul_b_out = sess.run(a_mul_b)
print_tf(a_mul_b_out)
```

```
TYPE IS
 <type 'numpy.float32'>
VALUE IS
 3.75
```

VARIABLES

```
weight = tf.Variable(tf.random_normal([5, 2], stddev=0.1))
print_tf(weight)
```

```
TYPE IS
 <class 'tensorflow.python.ops.variables.Variable'>
VALUE IS
 <tensorflow.python.ops.variables.Variable object at 0x7ff2bc04c
050>
```

```
weight_out = sess.run(weight)
print_tf(weight_out)
```

```
---------------------------------------------------------------------------

FailedPreconditionError                   Traceback (most recent
 call last)

<ipython-input-11-e453db2b7ada> in <module>()
----> 1 weight_out = sess.run(weight)
      2 print_tf(weight_out)


/usr/local/lib/python2.7/dist-packages/tensorflow/python/client/
session.pyc in run(self, fetches, feed_dict, options, run_metada
ta)
    338     try:
    339       result = self._run(None, fetches, feed_dict, optio
ns_ptr,
--> 340                          run_metadata_ptr)
    341       if run_metadata:
    342         proto_data = tf_session.TF_GetBuffer(run_metadat
a_ptr)


/usr/local/lib/python2.7/dist-packages/tensorflow/python/client/
session.pyc in _run(self, handle, fetches, feed_dict, options, r
un_metadata)
    562     try:
    563       results = self._do_run(handle, target_list, unique
_fetches,
--> 564                              feed_dict_string, options,
run_metadata)
    565     finally:
    566       # The movers are no longer used. Delete them.


/usr/local/lib/python2.7/dist-packages/tensorflow/python/client/
session.pyc in _do_run(self, handle, target_list, fetch_list, fe
ed_dict, options, run_metadata)
    635     if handle is None:
    636       return self._do_call(_run_fn, self._session, feed_
dict, fetch_list,
--> 637                            target_list, options, run_met
adata)
    638     else:
    639       return self._do_call(_prun_fn, self._session, hand
le, feed_dict,
```

```
/usr/local/lib/python2.7/dist-packages/tensorflow/python/client/
session.pyc in _do_call(self, fn, *args)
    657         # pylint: disable=protected-access
    658         raise errors._make_specific_exception(node_def, op
, error_message,
--> 659                                               e.code)
    660         # pylint: enable=protected-access
    661


FailedPreconditionError: Attempting to use uninitialized value V
ariable
     [[Node: Variable/_14 = _Send[T=DT_FLOAT, client_terminated=
false, recv_device="/job:localhost/replica:0/task:0/cpu:0", send
_device="/job:localhost/replica:0/task:0/gpu:0", send_device_inc
arnation=1, tensor_name="edge_25_Variable", _device="/job:localh
ost/replica:0/task:0/gpu:0"](Variable)]]
     [[Node: Variable/_15 = _Recv[_start_time=0, client_terminat
ed=false, recv_device="/job:localhost/replica:0/task:0/cpu:0", s
end_device="/job:localhost/replica:0/task:0/gpu:0", send_device_
incarnation=1, tensor_name="edge_25_Variable", tensor_type=DT_FL
OAT, _device="/job:localhost/replica:0/task:0/cpu:0"]()]]
```

WHY DOES THIS ERROR OCCURS?

```
init = tf.initialize_all_variables()
sess.run(init)
print ("INITIALIZING ALL VARIALBES")
```

```
INITIALIZING ALL VARIALBES
```

ONCE, WE INITIALIZE VARIABLES

```
weight_out = sess.run(weight)
print_tf(weight_out)
```

```
TYPE IS
 <type 'numpy.ndarray'>
VALUE IS
 [[ 0.08847231 -0.08040368]
 [-0.00344782 -0.32673332]
 [ 0.10427399 -0.12950435]
 [ 0.19032514 -0.1323577 ]
 [-0.00949183 -0.10073283]]
```

PLACEHOLDERS

```
x = tf.placeholder(tf.float32, [None, 5])
print_tf(x)
```

```
TYPE IS
 <class 'tensorflow.python.framework.ops.Tensor'>
VALUE IS
 Tensor("Placeholder:0", shape=(?, 5), dtype=float32)
```

OPERATION WITH VARIABLES AND PLACEHOLDERS

```
oper = tf.matmul(x, weight)
print_tf(oper)
```

```
TYPE IS
 <class 'tensorflow.python.framework.ops.Tensor'>
VALUE IS
 Tensor("MatMul_1:0", shape=(?, 2), dtype=float32)
```

```
data = np.random.rand(1, 5)
oper_out = sess.run(oper, feed_dict={x: data})
print_tf(oper_out)
```

```
TYPE IS
 <type 'numpy.ndarray'>
VALUE IS
 [[ 0.15189362 -0.18824303]]
```

```
data = np.random.rand(2, 5)
oper_out = sess.run(oper, feed_dict={x: data})
print_tf(oper_out)
```

```
TYPE IS
 <type 'numpy.ndarray'>
VALUE IS
 [[ 0.25157502 -0.50201333]
 [ 0.11929326 -0.4199847 ]]
```

MNIST

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data

%matplotlib inline
print ("packs loaded")
```

```
packs loaded
```

Download and Extract MNIST dataset

```
print ("Download and Extract MNIST dataset")
mnist = input_data.read_data_sets('data/', one_hot=True)
print
print (" tpye of 'mnist' is %s" % (type(mnist)))
print (" number of trian data is %d" % (mnist.train.num_examples
))
print (" number of test data is %d" % (mnist.test.num_examples))
```

```
Download and Extract MNIST dataset
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz

 tpye of 'mnist' is <class 'tensorflow.contrib.learn.python.lear
n.datasets.mnist.DataSets'>
 number of trian data is 55000
 number of test data is 10000
```

```
# What does the data of MNIST look like?
print ("What does the data of MNIST look like?")
trainimg   = mnist.train.images
trainlabel = mnist.train.labels
testimg    = mnist.test.images
testlabel  = mnist.test.labels
print
print (" type of 'trainimg' is %s"    % (type(trainimg)))
print (" type of 'trainlabel' is %s"  % (type(trainlabel)))
print (" type of 'testimg' is %s"     % (type(testimg)))
print (" type of 'testlabel' is %s"   % (type(testlabel)))
print (" shape of 'trainimg' is %s"   % (trainimg.shape,))
print (" shape of 'trainlabel' is %s" % (trainlabel.shape,))
print (" shape of 'testimg' is %s"    % (testimg.shape,))
print (" shape of 'testlabel' is %s"  % (testlabel.shape,))
```

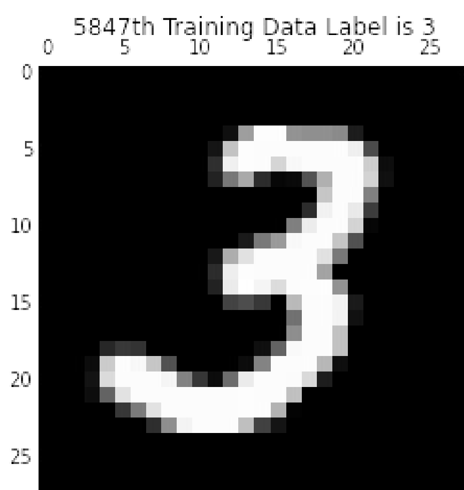
```
What does the data of MNIST look like?

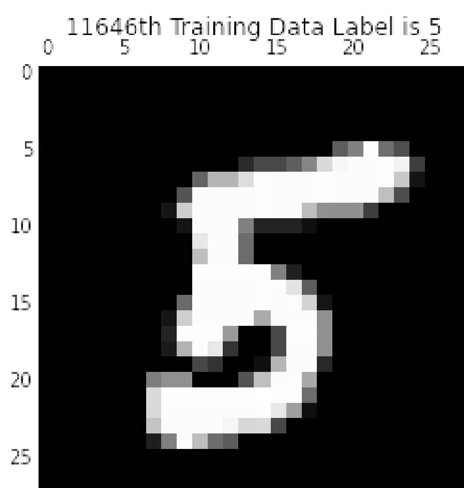
 type of 'trainimg' is <type 'numpy.ndarray'>
 type of 'trainlabel' is <type 'numpy.ndarray'>
 type of 'testimg' is <type 'numpy.ndarray'>
 type of 'testlabel' is <type 'numpy.ndarray'>
 shape of 'trainimg' is (55000, 784)
 shape of 'trainlabel' is (55000, 10)
 shape of 'testimg' is (10000, 784)
 shape of 'testlabel' is (10000, 10)
```

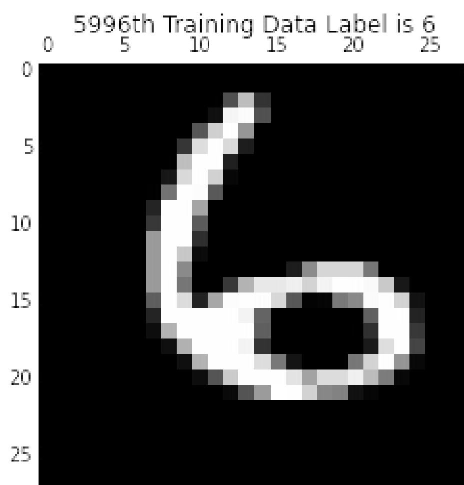
```
# How does the training data look like?
print ("How does the training data look like?")
nsample = 5
randidx = np.random.randint(trainimg.shape[0], size=nsample)

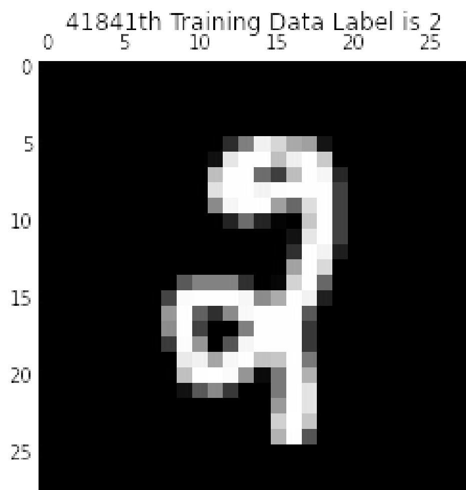
for i in randidx:
    curr_img   = np.reshape(trainimg[i, :], (28, 28)) # 28 by 28
 matrix
    curr_label = np.argmax(trainlabel[i, :] ) # Label
    plt.matshow(curr_img, cmap=plt.get_cmap('gray'))
    plt.title("" + str(i) + "th Training Data "
              + "Label is " + str(curr_label))
    print ("" + str(i) + "th Training Data "
           + "Label is " + str(curr_label))
```

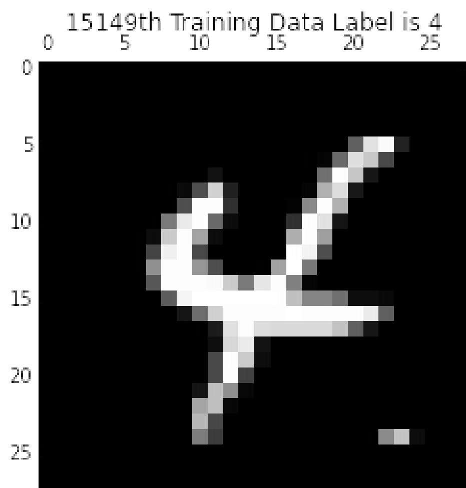
```
How does the training data look like?
5847th Training Data Label is 3
11646th Training Data Label is 5
5996th Training Data Label is 6
41841th Training Data Label is 2
15149th Training Data Label is 4
```












```
# Batch Learning?
print ("Batch Learning? ")
batch_size = 100
batch_xs, batch_ys = mnist.train.next_batch(batch_size)
print ("type of 'batch_xs' is %s" % (type(batch_xs)))
print ("type of 'batch_ys' is %s" % (type(batch_ys)))
print ("shape of 'batch_xs' is %s" % (batch_xs.shape,))
print ("shape of 'batch_ys' is %s" % (batch_ys.shape,))
```

```
Batch Learning?
type of 'batch_xs' is <type 'numpy.ndarray'>
type of 'batch_ys' is <type 'numpy.ndarray'>
shape of 'batch_xs' is (100, 784)
shape of 'batch_ys' is (100, 10)
```

```
# Get Random Batch with 'np.random.randint'
print ("5. Get Random Batch with 'np.random.randint'")
randidx   = np.random.randint(trainimg.shape[0], size=batch_size
)
batch_xs2 = trainimg[randidx, :]
batch_ys2 = trainlabel[randidx, :]
print ("type of 'batch_xs2' is %s" % (type(batch_xs2)))
print ("type of 'batch_ys2' is %s" % (type(batch_ys2)))
print ("shape of 'batch_xs2' is %s" % (batch_xs2.shape,))
print ("shape of 'batch_ys2' is %s" % (batch_ys2.shape,))
```

```
5. Get Random Batch with 'np.random.randint'
type of 'batch_xs2' is <type 'numpy.ndarray'>
type of 'batch_ys2' is <type 'numpy.ndarray'>
shape of 'batch_xs2' is (100, 784)
shape of 'batch_ys2' is (100, 10)
```

```
randidx
```

```
array([40597, 27095, 26480,  9962, 40322, 38562, 53100, 29354, 2
4853,
       39323, 12301, 29530,  6876, 17472, 11859, 32907, 31891, 4
3449,
       42376, 22173,   115, 16827, 47957, 10636, 43259, 16207, 3
3329,
       12654,  5640,  6254, 36093, 39494, 45642, 28959,  2347,
1911,
       11653, 40175, 28654, 29179, 36227,  3112, 35634, 22400, 3
8441,
       11548, 29659, 39165, 42957, 19418,  3168, 53571, 29323,
8976,
       18668, 46934, 19848, 19982,  9148, 16059, 48727, 31939, 1
1938,
       36669, 24021, 31104, 39372, 44231, 10097, 43418,  5704, 1
7825,
       54984, 38007,  2098, 31896, 41666,  8106, 37039,  3065, 4
4691,
        6446, 34830, 26956, 36618, 15762, 17894, 23088, 37065, 4
1814,
       20915, 13454, 43314,  2817, 34319, 13249, 39225, 36624, 5
3752, 44867])
```

LOAD PACKAGES!

```
import numpy as np
print ("Loading package(s)")
```

```
Loading package(s)
```

PRINT function usages

```
print ("Hello, world")

# THERE ARE THREE POPULAR TYPES
# 1. INTEGER
x = 3;
print ("Integer: %01d, %02d, %03d, %04d, %05d"
       % (x, x, x, x, x))
# 2. FLOAT
x = 123.456;
print ("Float: %.0f, %.1f, %.2f, %1.2f, %2.2f"
       % (x, x, x, x, x))
# 3. STRING
x = "Hello, world"
print ("String: [%s], [%3s], [%20s]"
       % (x, x, x))
```

```
Hello, world
Integer: 3, 03, 003, 0003, 00003
Float: 123, 123.5, 123.46, 123.46, 123.46
String: [Hello, world], [Hello, world], [        Hello, world]
```

FOR + IF/ELSE

```
dlmethods = ["ANN", "MLP", "CNN", "RNN", "DAE"]

for alg in dlmethods:
    if alg in ["ANN", "MLP"]:
        print ("We have seen %s" % (alg))
```

```
We have seen ANN
We have seen MLP
```

```
dlmethods = ["ANN", "MLP", "CNN", "RNN", "DAE"];
for alg in dlmethods:
    if alg in ["ANN", "MLP", "CNN"]:
        print ("%s is a feed-forward network." % (alg))
    elif alg in ["RNN"]:
        print ("%s is a recurrent network." % (alg))
    else:
        print ("%s is an unsupervised method." % (alg))

# Little more advanced?
print("\nFOR loop with index.")
for alg, i in zip(dlmethods, range(len(dlmethods))):
    if alg in ["ANN", "MLP", "CNN"]:
        print ("[%d/%d] %s is a feed-forward network."
               % (i, len(dlmethods), alg))
    elif alg in ["RNN"]:
        print ("[%d/%d] %s is a recurrent network."
               % (i, len(dlmethods), alg))
    else:
        print ("[%d/%d] %s is an unsupervised method."
               % (i, len(dlmethods), alg))
```

Note that, index starts with 0 !

Let's make a function in Python

```
# Function definition looks like this
def sum(a, b):
    return a+b
X = 10.
Y = 20.
# Usage
print ("%.1f + %.1f = %.1f" % (X, Y, sum(X, Y)))
```

```
10.0 + 20.0 = 30.0
```

String operations

```
head = "Deep learning"
body = "very "
tail = "HARD."
print (head + " is " + body + tail)

# Repeat words
print (head + " is " + body*3 + tail)
print (head + " is " + body*10 + tail)

# It is used in this way
print ("\n" + "="*50)
print (" "*15 + "It is used in this way")
print ("="*50 + "\n")

# Indexing characters in the string
x = "Hello, world"
for i in range(len(x)):
    print ("Index: [%02d/%02d] Char: %s"
           % (i, len(x), x[i]))
```

```
Deep learning is very HARD.
Deep learning is very very very HARD.
Deep learning is very very very very very very very very very ve
ry HARD.

==================================================
               It is used in this way
==================================================

Index: [00/12] Char: H
Index: [01/12] Char: e
Index: [02/12] Char: l
Index: [03/12] Char: l
Index: [04/12] Char: o
Index: [05/12] Char: ,
Index: [06/12] Char:  
Index: [07/12] Char: w
Index: [08/12] Char: o
Index: [09/12] Char: r
Index: [10/12] Char: l
Index: [11/12] Char: d
```

```
# More indexing
print ""
idx = -2
print ("(%d)th char is %s" % (idx, x[idx]))
idxfr = 0
idxto = 8
print ("String from %d to %d is [%s]"
       % (idxfr, idxto, x[idxfr:idxto]))
idxfr = 4
print ("String from %d to END is [%s]"
       % (idxfr, x[idxfr:]))
x = "20160607Cloudy"
year = x[:4]
day = x[4:8]
weather = x[8:]
print ("[%s] -> [%s] + [%s] + [%s] "
       % (x, year, day, weather))
```

```
(-2)th char is l
String from 0 to 8 is [Hello, w]
String from 4 to END is [o, world]
[20160607Cloudy] -> [2016] + [0607] + [Cloudy]
```

LIST

```
a = []
b = [1, 2, 3]
c = ["Hello", ",", "world"]
d = [1, 2, 3, "x", "y", "z"]
x = []
print x
x.append('a')
print x
x.append(123)
print x
x.append(["a", "b"])
print x
print ("Length of x is %d "
       % (len(x)))
for i in range(len(x)):
    print ("[%02d/%02d] %s"
           % (i, len(x), x[i]))
```

```
[]
['a']
['a', 123]
['a', 123, ['a', 'b']]
Length of x is 3
[00/03] a
[01/03] 123
[02/03] ['a', 'b']
```

```
z = []
z.append(1)
z.append(2)
z.append(3)
z.append('Hello')
for i in range(len(z)):
    print (z[i])
```

```
1
2
3
Hello
```

DICTIONARY

```
dic = dict()
dic["name"] = "Sungjoon"
dic["age"] = 31
dic["job"] = "Ph.D. Candidate"

print dic
```

```
{'job': 'Ph.D. Candidate', 'age': 31, 'name': 'Sungjoon'}
```

Class

```
class Greeter:

    # Constructor
    def __init__(self, name):
        self.name = name  # Create an instance variable

    # Instance method
    def greet(self, loud=False):
        if loud:
            print ('HELLO, %s!'
                   % self.name.upper())
        else:
            print ('Hello, %s'
                   % self.name)

g = Greeter('Fred')  # Construct an instance of the Greeter class

g.greet()            # Call an instance method; prints "Hello, F
red"
g.greet(loud=True)   # Call an instance method; prints "HELLO, F
RED!"
```

```
Hello, Fred
HELLO, FRED!
```

```
def print_np(x):
    print ("Type is %s" % (type(x)))
    print ("Shape is %s" % (x.shape,))
    print ("Values are: \n%s" % (x))
    print
```

RANK 1 ARRAY

```
x = np.array([1, 2, 3]) # rank 1 array
print_np(x)

x[0] = 5
print_np(x)
```

```
Type is <type 'numpy.ndarray'>
Shape is (3,)
Values are:
[1 2 3]

Type is <type 'numpy.ndarray'>
Shape is (3,)
Values are:
[5 2 3]
```

RANK 2 ARRAY

```
y = np.array([[1,2,3], [4,5,6]])
print_np(y)
```

```
Type is <type 'numpy.ndarray'>
Shape is (2, 3)
Values are:
[[1 2 3]
 [4 5 6]]
```

ZEROS

```
a = np.zeros((3, 2))
print_np(a)
```

```
Type is <type 'numpy.ndarray'>
Shape is (3, 2)
Values are:
[[ 0.  0.]
 [ 0.  0.]
 [ 0.  0.]]
```

ONES

```
b = np.ones((1, 2))
print_np(b)
```

```
Type is <type 'numpy.ndarray'>
Shape is (1, 2)
Values are:
[[ 1.  1.]]
```

IDENTITY

```
c = np.eye(2, 2)
print_np(c)
```

```
Type is <type 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[ 1.  0.]
 [ 0.  1.]]
```

RANDOM (UNIFORM)

```
d = np.random.random((2, 2))
print_np(d)
```

```
Type is <type 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[ 0.09677829  0.13234216]
 [ 0.87168847  0.63200027]]
```

RANDOM (GAUSSIAN)

```
e = np.random.randn(1, 10)
print_np(e)
```

```
Type is <type 'numpy.ndarray'>
Shape is (1, 10)
Values are:
[[ 1.12732237 -1.50937817 -0.01637454  0.02860102  0.2353765   0
.36251934
  -1.30868695 -1.16874378  0.8219648  -0.99443059]]
```

ARRAY INDEXING

```
# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print_np(a)

print
# Use slicing to pull out the subarray consisting
# of the first 2 rows
# and columns 1 and 2; b is the following array
# of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]
print_np(b)
```

GET ROW

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print_np(a)

row_r1 = a[1, :]    # Rank 1 view of the second row of a
row_r2 = a[1:2, :]  # Rank 2 view of the second row of a
row_r3 = a[[1], :]  # Rank 2 view of the second row of a

print_np(row_r1)
print_np(row_r2)
print_np(row_r3)
```

```
a = np.array([[1,2], [3, 4], [5, 6]])
print_np(a)

# An example of integer array indexing.
# The returned array will have shape (3,) and
b = a[[0, 1, 2], [0, 1, 0]]
print_np(b)

# The above example of integer array indexing
# is equivalent to this:
c = np.array([a[0, 0], a[1, 1], a[2, 0]])
print_np(c)
```

DATATYPES

```
x = np.array([1, 2])  # Let numpy choose the datatype
y = np.array([1.0, 2.0])  # Let numpy choose the datatype
z = np.array([1, 2], dtype=np.int64)  # particular datatype

print_np(x)
print_np(y)
print_np(z)
```

Array math

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array
print x + y
print np.add(x, y)
```

```
[[  6.   8.]
 [ 10.  12.]]
[[  6.   8.]
 [ 10.  12.]]
```

```
#  Elementwise difference; both produce the array
print x - y
print np.subtract(x, y)
```

```
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
```

```
# Elementwise product; both produce the array
print x * y
print np.multiply(x, y)
```

```
[[  5.  12.]
 [ 21.  32.]]
[[  5.  12.]
 [ 21.  32.]]
```

```
# Elementwise division; both produce the array
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
print x / y
print np.divide(x, y)
```

```
[[ 0.2         0.33333333]
 [ 0.42857143  0.5       ]]
[[ 0.2         0.33333333]
 [ 0.42857143  0.5       ]]
```

```
# Elementwise square root; produces the array
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print np.sqrt(x)
```

```
[[ 1.          1.41421356]
 [ 1.73205081  2.        ]]
```

```
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])
v = np.array([9,10])
w = np.array([11, 12])

print_np(x)
print_np(y)
print_np(v)
print_np(w)

# Inner product of vectors; both produce 219
print v.dot(w)
print np.dot(v, w) # <= v * w'
# Matrix / vector product; both produce the rank 1 array [29 67]
print x.dot(v)
print np.dot(x, v) # <= x * v'
# Matrix / matrix product; both produce the rank 2 array
# [[19 22]
#  [43 50]]
print x.dot(y)
print np.dot(x, y)
```

```
x = np.array([[1,2],[3,4]])
print_np(x)
print
print x
print x.T
print np.sum(x)  # Compute sum of all elements
print np.sum(x, axis=0)  # Compute sum of each column
print np.sum(x, axis=1)  # Compute sum of each row
```

```
print x
print x.T
```

```
v = np.array([1,2,3])
print v
print v.T
```

```
v = np.array([[1,2,3]])
print v
print v.T
```

Other useful operations

```
# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x)    # Create an empty matrix
                        # with the same shape as x

print_np(x)
print_np(v)
print_np(y)
```

```
# Add the vector v to each row of the matrix x
# with an explicit loop
for i in range(4):
    y[i, :] = x[i, :] + v
print_np(y)
```

```
vv = np.tile(v, (4, 1))  # Stack 4 copies of v on top of each ot
her
print_np(vv)             # Prints "[[1 0 1]
                         #          [1 0 1]
                         #          [1 0 1]
                         #          [1 0 1]]"
```

```
# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v  # Add v to each row of x using BROADCASTING
print_np(x)
print_np(v)
print_np(y)
```

```
# Add a vector to each row of a matrix
x = np.array([[1,2,3], [4,5,6]])

print_np(x)
print_np(v)
print x + v
```

```
# Add a vector to each column of a matrix
print_np(x)
print_np(w)
print (x.T + w).T

# Another solution is to reshape w
# to be a row vector of shape (2, 1);
print
print x + np.reshape(w, (2, 1))
```

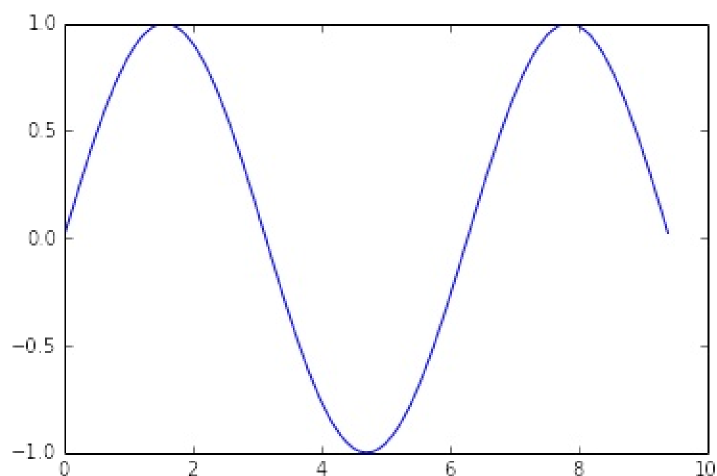
Matplotlib

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

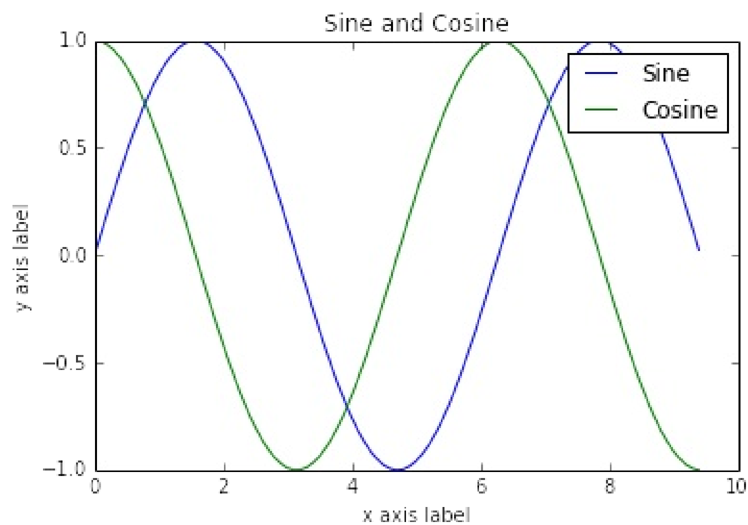
# Plot the points using matplotlib
plt.plot(x, y)
```

```
[<matplotlib.lines.Line2D at 0x7f7bc21b91d0>]
```



```
y_sin = np.sin(x)
y_cos = np.cos(x)
# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])

# Show the figure.
plt.show()
```



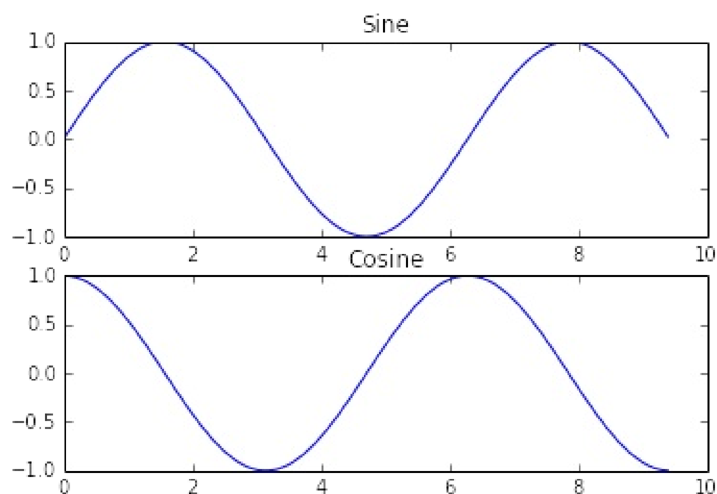
```
# Compute the x and y coordinates for points
# on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```



```
"""
 Basic imgae load, plot, resize, etc..
 Sungjoon Choi (sungjoon.choi@cpslab.snu.ac.kr)
"""
# Import packs
import numpy as np
import os
from scipy.misc import imread, imresize
import matplotlib.pyplot as plt
import skimage.io
import skimage.transform
# import tensorflow as tf

%matplotlib inline

print ("Packs loaded")
```

```
Packs loaded
```

```
# Print Current Folder
cwd = os.getcwd()
print ("Current folder is %s" % (cwd) )

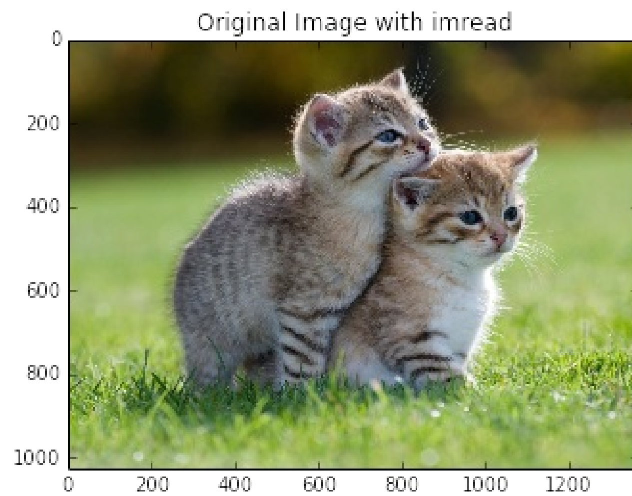
# Useful function
def print_typeshape(img):
    print("Type is %s" % (type(img)))
    print("Shape is %s" % (img.shape,))
```

```
Current folder is /home/enginius/github/tensorflow-101/notebooks
```

Load & plot

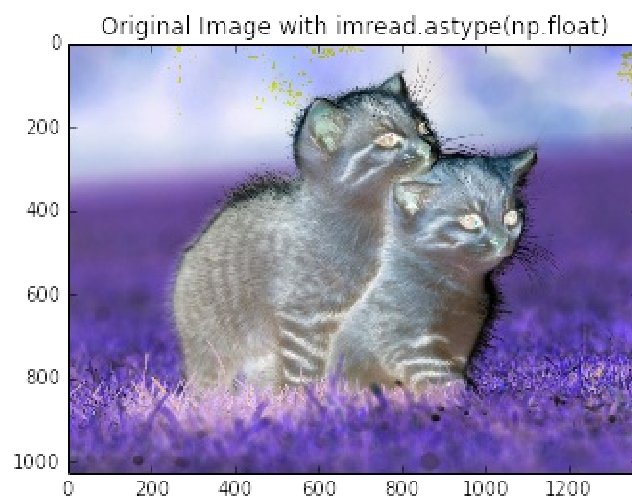
```
# Load
cat = imread(cwd + "/images/cat.jpg")
print_typeshape(cat)
# Plot
plt.figure(0)
plt.imshow(cat)
plt.title("Original Image with imread")
plt.draw()
```

```
Type is <type 'numpy.ndarray'>
Shape is (1026, 1368, 3)
```



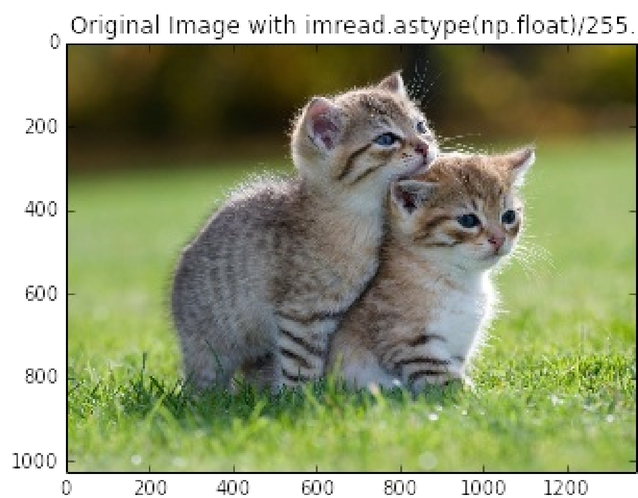

```
# Load
cat2 = imread(cwd + "/images/cat.jpg").astype(np.float)
print_typeshape(cat2)
# Plot
plt.figure(0)
plt.imshow(cat2)
plt.title("Original Image with imread.astype(np.float)")
plt.draw()
```

```
Type is <type 'numpy.ndarray'>
Shape is (1026, 1368, 3)
```



```
# Load
cat3 = imread(cwd + "/images/cat.jpg").astype(np.float)
print_typeshape(cat3)
# Plot
plt.figure(0)
plt.imshow(cat3/255.)
plt.title("Original Image with imread.astype(np.float)/255.")
plt.draw()
```

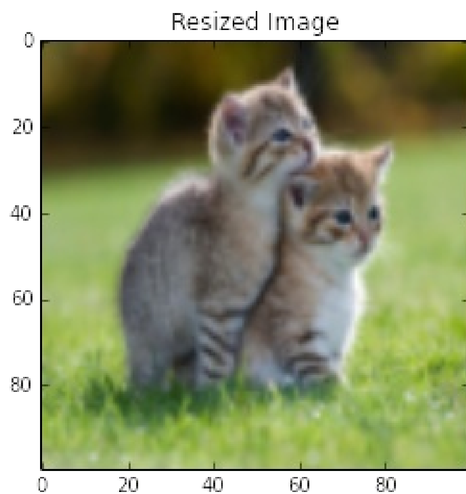
```
Type is <type 'numpy.ndarray'>
Shape is (1026, 1368, 3)
```



Resize

```
# Resize
catsmall = imresize(cat, [100, 100, 3])
print_typeshape(catsmall)
# Plot
plt.figure(1)
plt.imshow(catsmall)
plt.title("Resized Image")
plt.draw()
```

```
Type is <type 'numpy.ndarray'>
Shape is (100, 100, 3)
```



Grayscale

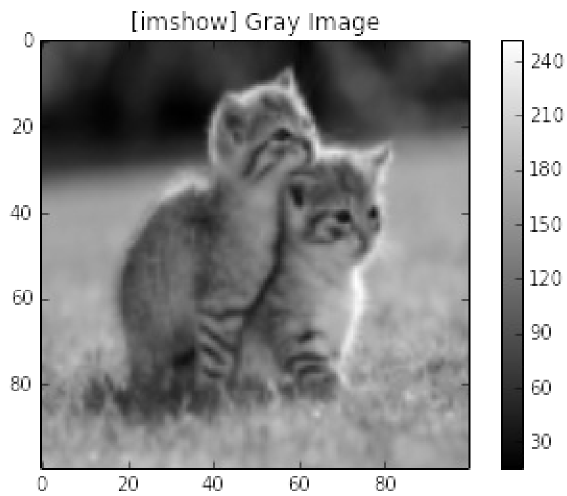
```
# Grayscale
def rgb2gray(rgb):
    if len(rgb.shape) is 3:
        return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])
    else:
        print ("Current Image if GRAY!")
        return rgb
catsmallgray = rgb2gray(catsmall)

print ("size of catsmallgray is %s" % (catsmallgray.shape,))
print ("type of catsmallgray is", type(catsmallgray))

plt.figure(2)
plt.imshow(catsmallgray, cmap=plt.get_cmap("gray"))
plt.title("[imshow] Gray Image")
plt.colorbar()
plt.draw()
```

```
size of catsmallgray is (100, 100)
('type of catsmallgray is', <type 'numpy.ndarray'>)


/usr/lib/pymodules/python2.7/matplotlib/collections.py:548: Futu
reWarning: elementwise comparison failed; returning scalar inste
ad, but in the future will perform elementwise comparison
  if self._edgecolors == 'face':
```



Reshape

```
# Convert to Vector
catrowvec = np.reshape(catsmallgray, (1, -1));
print ("size of catrowvec is %s" % (catrowvec.shape,))
print ("type of catrowvec is", type(catrowvec))

# Convert to Matrix
catmatrix = np.reshape(catrowvec, (100, 100));
print ("size of catmatrix is %s" % (catmatrix.shape,))
print ("type of catmatrix is", type(catmatrix))
```

```
size of catrowvec is (1, 10000)
('type of catrowvec is', <type 'numpy.ndarray'>)
size of catmatrix is (100, 100)
('type of catmatrix is', <type 'numpy.ndarray'>)
```

Load from folder

```
# Load from Folder
cwd  = os.getcwd()
path = cwd + "/images/cats"
valid_exts = [".jpg",".gif",".png",".tga", ".jpeg"]

# print ("Images in %s are: \n %s" % (path, os.listdir(path)))
print ("%d files in %s" % (len(os.listdir(path)), path))

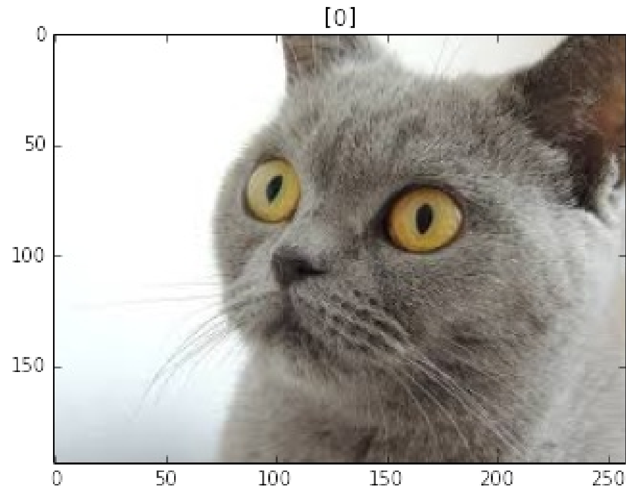
# Append Images and their Names to Lists
imgs = []
names = []
for f in os.listdir(path):
    # For all files
    ext = os.path.splitext(f)[1]
    # Check types
    if ext.lower() not in valid_exts:
        continue
    fullpath = os.path.join(path,f)
    imgs.append(imread(fullpath))
    names.append(os.path.splitext(f)[0]+os.path.splitext(f)[1])
print ("%d images loaded" % (len(imgs)))
```

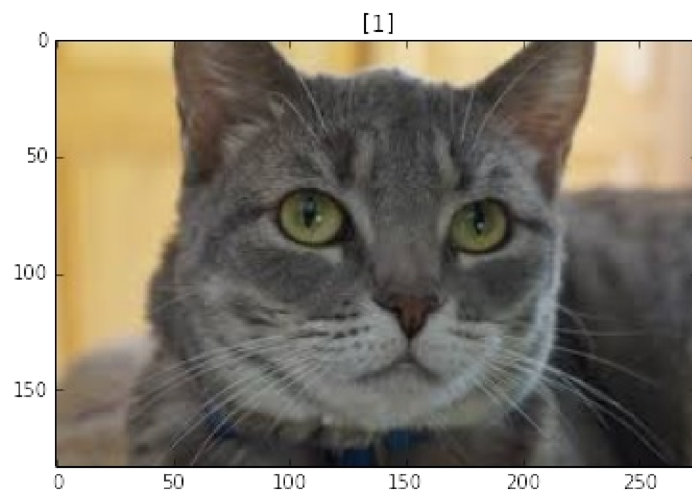
```
38 files in /home/enginius/github/tensorflow-101/notebooks/image
s/cats
38 images loaded
```

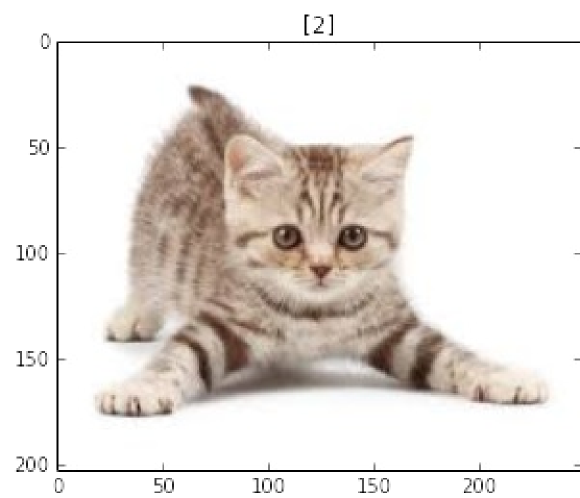
```
# Check
nimgs = len(imgs)
randidx = np.sort(np.random.randint(nimgs, size=3))
print ("Type of 'imgs': ", type(imgs))
print ("Length of 'imgs': ", len(imgs))
for curr_img, curr_name, i \
    in zip([imgs[j] for j in randidx]
           , [names[j] for j in randidx]
           , range(len(randidx))):
    print ("[%d] Type of 'curr_img': %s" % (i, type(curr_img)))
    print ("    Name is: %s" % (curr_name))
    print ("    Size of 'curr_img': %s" % (curr_img.shape,))
```

```
("Type of 'imgs': ", <type 'list'>)
("Length of 'imgs': ", 38)
[0] Type of 'curr_img': <type 'numpy.ndarray'>
    Name is: images (19).jpeg
    Size of 'curr_img': (246, 205, 3)
[1] Type of 'curr_img': <type 'numpy.ndarray'>
    Name is: images (32).jpeg
    Size of 'curr_img': (194, 259, 3)
[2] Type of 'curr_img': <type 'numpy.ndarray'>
    Name is: images (11).jpeg
    Size of 'curr_img': (183, 275, 3)
```

```
# Plot Images in 'imgs' list
nimgs = len(imgs)
randidx = np.sort(np.random.randint(nimgs, size=3))
for curr_img, curr_name, i \
    in zip([imgs[j] for j in randidx]
           , [names[j] for j in randidx], range(len(randidx))):
    plt.figure(i)
    plt.imshow(curr_img)
    plt.title("[" + str(i) + "] ")
    plt.draw()
```








```
print "That was all!"
```

```
That was all!
```

Basic data set generation

```
import numpy as np
import os
from scipy.misc import imread, imresize
import matplotlib.pyplot as plt
%matplotlib inline
print ("Package loaded")
cwd = os.getcwd()
print ("Current folder is %s" % (cwd) )
```

```
Package loaded
Current folder is /home/enginius/github/tensorflow-101/notebooks
```

SPECIFY THE FOLDER PATHS

+ RESHAPE SIZE + GRAYSCALE

```
# Training set folder
paths = {"../../img_dataset/celebs/Arnold_Schwarzenegger"
        , "../../img_dataset/celebs/Junichiro_Koizumi"
        , "../../img_dataset/celebs/Vladimir_Putin"
        , "../../img_dataset/celebs/George_W_Bush"}
# The reshape size
imgsize = [64, 64]
# Grayscale
use_gray = 1
# Save name
data_name = "custom_data"

print ("Your images should be at")
for i, path in enumerate(paths):
    print (" [%d/%d] %s/%s" % (i, len(paths), cwd, path))

print ("Data will be saved to %s"
       % (cwd + '/data/' + data_name + '.npz'))
```

```
Your images should be at
 [0/4] /home/enginius/github/tensorflow-101/notebooks/../../img_
dataset/celebs/George_W_Bush
 [1/4] /home/enginius/github/tensorflow-101/notebooks/../../img_
dataset/celebs/Arnold_Schwarzenegger
 [2/4] /home/enginius/github/tensorflow-101/notebooks/../../img_
dataset/celebs/Junichiro_Koizumi
 [3/4] /home/enginius/github/tensorflow-101/notebooks/../../img_
dataset/celebs/Vladimir_Putin
Data will be saved to /home/enginius/github/tensorflow-101/noteb
ooks/data/custom_data.npz
```

RGB 2 GRAY FUNCTION

```
def rgb2gray(rgb):
    if len(rgb.shape) is 3:
        return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])
    else:
        # print ("Current Image if GRAY!")
        return rgb
```

LOAD IMAGES

```
nclass      = len(paths)
valid_exts = [".jpg",".gif",".png",".tga", ".jpeg"]
imgcnt      = 0
for i, relpath in zip(range(nclass), paths):
    path = cwd + "/" + relpath
    flist = os.listdir(path)
    for f in flist:
        if os.path.splitext(f)[1].lower() not in valid_exts:
            continue
        fullpath = os.path.join(path, f)
        currimg  = imread(fullpath)
        # Convert to grayscale
        if use_gray:
            grayimg  = rgb2gray(currimg)
        else:
            grayimg  = currimg
        # Reshape
        graysmall = imresize(grayimg, [imgsize[0], imgsize[1]])/
255.
        grayvec   = np.reshape(graysmall, (1, -1))
        # Save
        curr_label = np.eye(nclass, nclass)[i:i+1, :]
        if imgcnt is 0:
            totalimg   = grayvec
            totallabel = curr_label
        else:
            totalimg   = np.concatenate((totalimg, grayvec), axi
s=0)
            totallabel = np.concatenate((totallabel, curr_label)
, axis=0)
        imgcnt    = imgcnt + 1
print ("Total %d images loaded." % (imgcnt))
```

```
Total 681 images loaded.
```

DIVIDE TOTAL DATA INTO TRAINING AND TEST SET

```
def print_shape(string, x):
    print ("Shape of '%s' is %s" % (string, x.shape,))

randidx    = np.random.randint(imgcnt, size=imgcnt)
trainidx   = randidx[0:int(3*imgcnt/5)]
testidx    = randidx[int(3*imgcnt/5):imgcnt]
trainimg   = totalimg[trainidx, :]
trainlabel = totallabel[trainidx, :]
testimg    = totalimg[testidx, :]
testlabel  = totallabel[testidx, :]
print_shape("trainimg", trainimg)
print_shape("trainlabel", trainlabel)
print_shape("testimg", testimg)
print_shape("testlabel", testlabel)
```

```
Shape of 'trainimg' is (408, 4096)
Shape of 'trainlabel' is (408, 4)
Shape of 'testimg' is (273, 4096)
Shape of 'testlabel' is (273, 4)
```

SAVE TO NPZ

```
savepath = cwd + "/data/" + data_name + ".npz"
np.savez(savepath, trainimg=trainimg, trainlabel=trainlabel
         , testimg=testimg, testlabel=testlabel, imgsize=imgsize
, use_gray=use_gray)
print ("Saved to %s" % (savepath))
```

```
Saved to /home/enginius/github/tensorflow-101/notebooks/data/cus
tom_data.npz
```

LOAD TO CHECK!

```
# Load them!
cwd = os.getcwd()
loadpath = cwd + "/data/" + data_name + ".npz"
l = np.load(loadpath)

# See what's in here
l.files

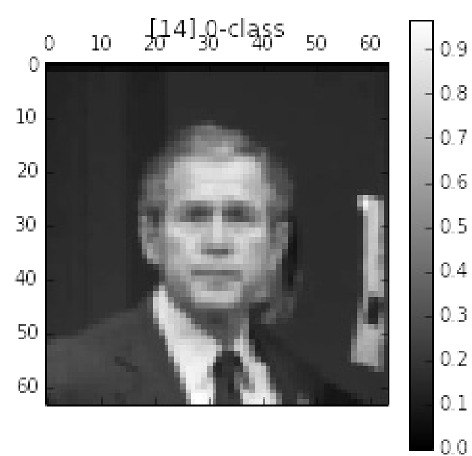
# Parse data
trainimg_loaded = l['trainimg']
trainlabel_loaded = l['trainlabel']
testimg_loaded = l['testimg']
testlabel_loaded = l['testlabel']

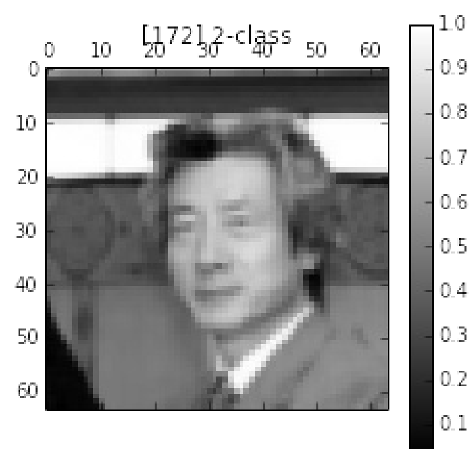
print ("%d train images loaded" % (trainimg_loaded.shape[0]))
print ("%d test images loaded" % (testimg_loaded.shape[0]))
print ("Loaded from to %s" % (savepath))
```

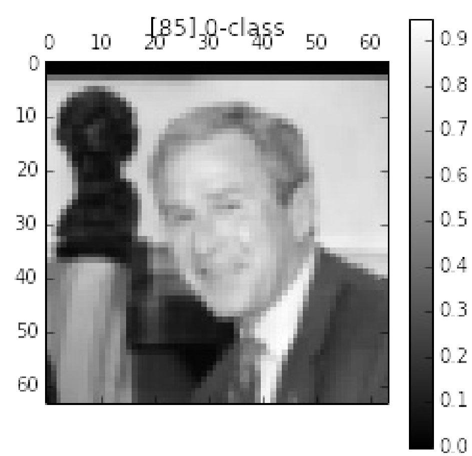
```
408 train images loaded
273 test images loaded
Loaded from to /home/enginius/github/tensorflow-101/notebooks/da
ta/custom_data.npz
```

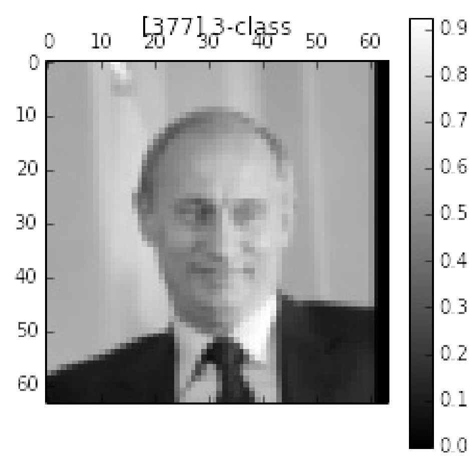
PLOT RANDOMLY SELECTED TRAIN IMAGES

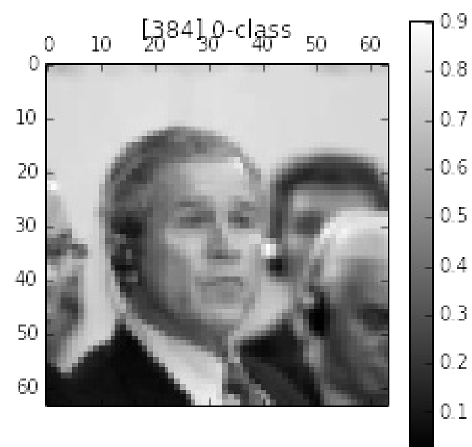
```
ntrain_loaded = trainimg_loaded.shape[0]
batch_size = 10;
randidx = np.random.randint(ntrain_loaded, size=batch_size)
for i in randidx:
    currimg = np.reshape(trainimg_loaded[i, :], (imgsize[0], -1)
)
    currlabel_onehot = trainlabel_loaded[i, :]
    currlabel = np.argmax(currlabel_onehot)
    if use_gray:
        currimg = np.reshape(trainimg[i, :], (imgsize[0], -1))
        plt.matshow(currimg, cmap=plt.get_cmap('gray'))
        plt.colorbar()
    else:
        currimg = np.reshape(trainimg[i, :], (imgsize[0], imgsiz
e[1], 3))
        plt.imshow(currimg)
    title_string = "[%d] %d-class" % (i, currlabel)
    plt.title(title_string)
    plt.show()
```

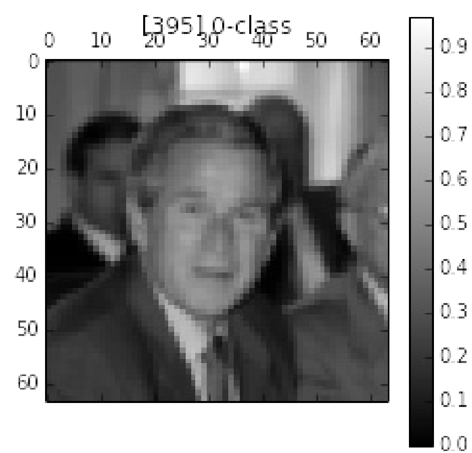

[14] 0-class

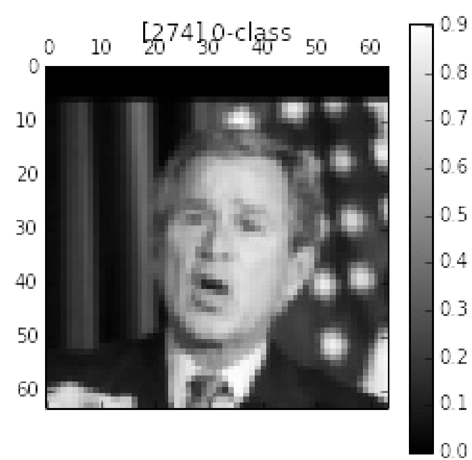

[172] 2-class

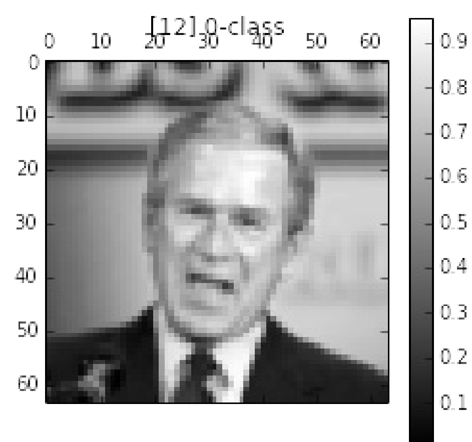

[85] 0-class

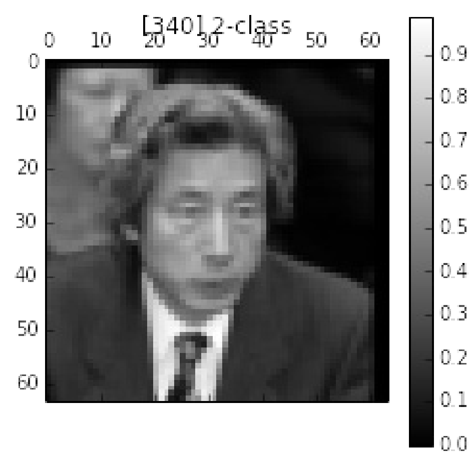

[377] 3-class
0 10 20 30 40 50 60
0 10 20 30 40 50 60
0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1 0.0

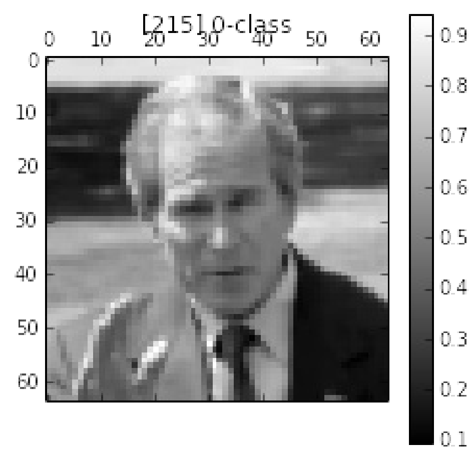

[384] 0-class
0 10 20 30 40 50 60
0 10 20 30 40 50 60
0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1


[395] 0-class
0 10 20 30 40 50 60
0 10 20 30 40 50 60
0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1 0.0


[274] 0-class


[12] 0-class

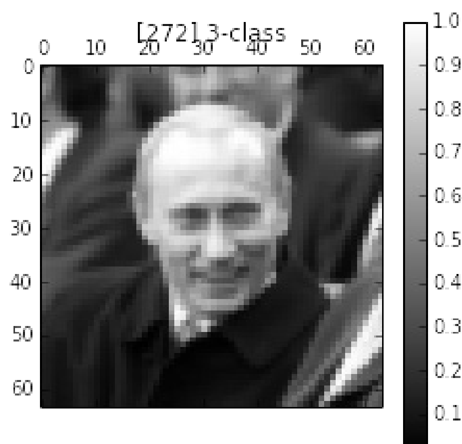

[340] 2-class

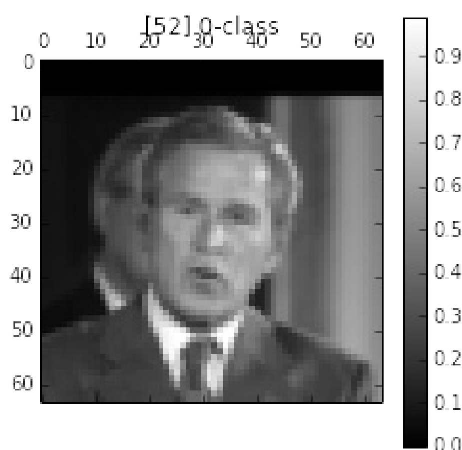

[215] 0-class

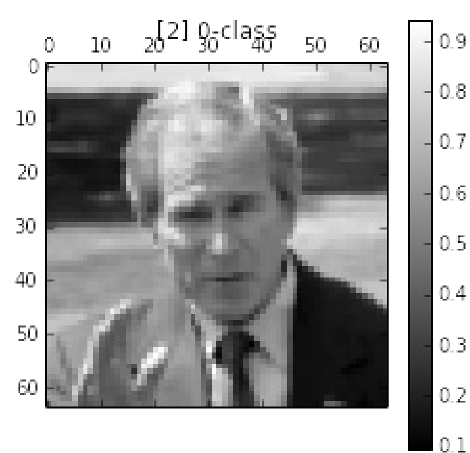
PLOT RANDOMLY SELECTED TEST IMAGES

```
# Do batch stuff using loaded data
ntest_loaded = testimg_loaded.shape[0]
batch_size   = 3;
randidx      = np.random.randint(ntest_loaded, size=batch_size)
for i in randidx:
    currimg = np.reshape(testimg_loaded[i, :], (imgsize[0], -1))
    currlabel_onehot = testlabel_loaded[i, :]
    currlabel = np.argmax(currlabel_onehot)

    if use_gray:
        currimg = np.reshape(testimg[i, :], (imgsize[0], -1))
        plt.matshow(currimg, cmap=plt.get_cmap('gray'))
        plt.colorbar()
    else:
        currimg = np.reshape(testimg[i, :], (imgsize[0], imgsize[
1], 3))
        plt.imshow(currimg)
    title_string = "[%d] %d-class" % (i, currlabel)
    plt.title(title_string)
    plt.show()
```






[2] 0-class
0
10
20
30
40
50
60
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

Machine Learing Basics with TensorFlow

```
"""
Linear Regression with TensorFlow
sungjoon.choi@cpslab.snu.ac.kr
"""
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
print ("Packages Loaded")
```

```
Packages Loaded
```

```
# Generate training data
np.random.seed(1)
def f(x, a, b):
    n    = train_X.size
    vals = np.zeros((1, n))
    for i in range(0, n):
        ax  = np.multiply(a, x.item(i))
        val = np.add(ax, b)
        vals[0, i] = val
    return vals

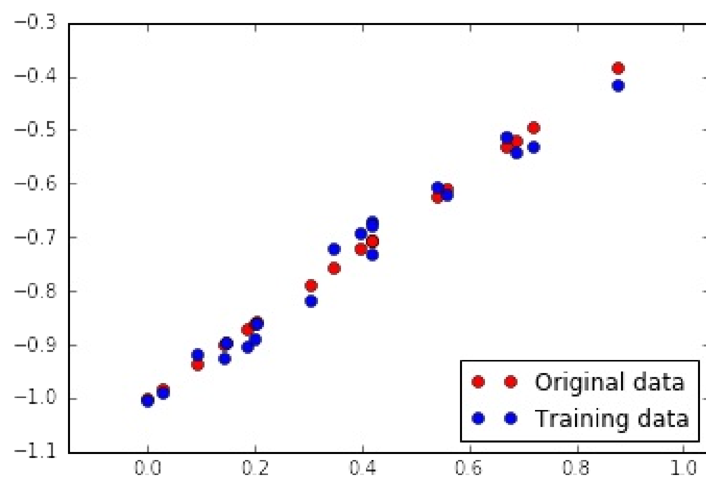
Wref = 0.7
bref = -1.
n    = 20
noise_var = 0.001
train_X   = np.random.random((1, n))
ref_Y     = f(train_X, Wref, bref)
train_Y   = ref_Y + np.sqrt(noise_var)*np.random.randn(1, n)
n_samples = train_X.size # <= Just for using size operator
print ("")
print (" Type of 'train_X' is ", type(train_X))
print (" Shape of 'train_X' is %s" % (train_X.shape,))
print (" Type of 'train_Y' is ", type(train_Y))
print (" Shape of 'train_Y' is %s" % (train_Y.shape,))

# Plot
plt.figure(1)
plt.plot(train_X[0, :], ref_Y[0, :], 'ro', label='Original data'
)
plt.plot(train_X[0, :], train_Y[0, :], 'bo', label='Training dat
a')
plt.axis('equal')
plt.legend(loc='lower right')
```

```
Type of 'train_X' is  <class 'numpy.ndarray'>
Shape of 'train_X' is (1, 20)
Type of 'train_Y' is  <class 'numpy.ndarray'>
Shape of 'train_Y' is (1, 20)




<matplotlib.legend.Legend at 0x7ff521bd5f60>
```



```
# Prepare for Linear Regression

# Parameters
training_epochs = 2000
display_step    = 50


# Set TensorFlow Graph
X = tf.placeholder(tf.float32, name="input")
Y = tf.placeholder(tf.float32, name="output")
W = tf.Variable(np.random.randn(), name="weight")
b = tf.Variable(np.random.randn(), name="bias")

# Construct a Model
activation = tf.add(tf.mul(X, W), b)

# Define Error Measure and Optimizer
learning_rate   = 0.01
cost = tf.reduce_mean(tf.pow(activation-Y, 2))
# learning_rate   = 0.001
# cost = tf.sqrt(tf.reduce_sum(tf.pow(activation-Y, 2)))
optimizer = tf.train.GradientDescentOptimizer(learning_rate).min
imize(cost) #Gradient descent

"""
 tf.reduce_sum()
 tf.reduce_mean()
 _____

 tf.pow(Yhat, Y, 2)
 tf.nn.softmax_cross_entropy_with_logits(Yhat, Y)
 _____

 tf.train.GradientDescentOptimizer(0.05).minimize(cost)
 tf.train.RMSPropOptimizer(0.001, 0.9).minimize(cost)
"""

# Initializer
init = tf.initialize_all_variables()
```

```
# Run!
sess = tf.Session()
# Initialize
sess.run(init)
for epoch in range(training_epochs):
    for (x, y) in zip(train_X[0, :], train_Y[0, :]):
        # print "x: ", x, " y: ", y
        sess.run(optimizer, feed_dict={X:x, Y:y})

    # Check cost
    if epoch % display_step == 0:
        costval = sess.run(cost, feed_dict={X: train_X, Y:train_
Y})
        print ("Epoch:", "%04d"%(epoch+1), "cost=", "{:.5f}".for
mat(costval))
        Wtemp = sess.run(W)
        btemp = sess.run(b)
        print (" Wtemp is", "{:.4f}".format(Wtemp), "btemp is",
"{:.4f}".format(btemp))
        print (" Wref is", "{:.4f}".format(Wref), "bref is", "{:
.4f}".format(bref))

# Final W and b
Wopt = sess.run(W)
bopt = sess.run(b)
fopt = f(train_X, Wopt, bopt)
```

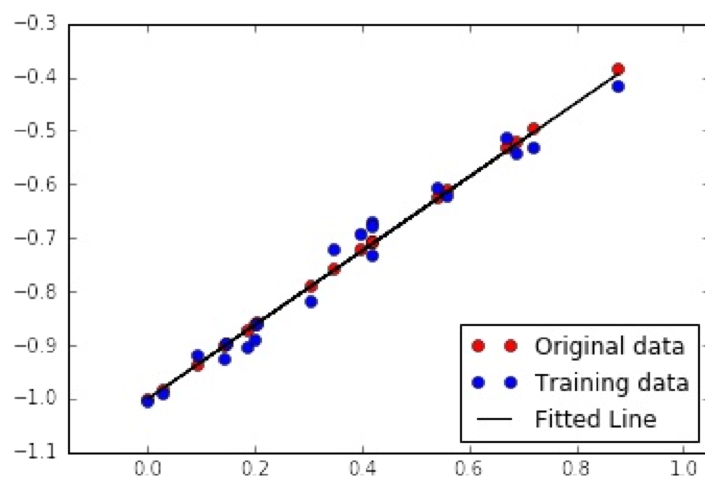
```
Epoch: 0001 cost= 0.21532
 Wtemp is -0.6964 btemp is -0.1715
 Wref is 0.7000 bref is -1.0000
Epoch: 0051 cost= 0.01696
 Wtemp is 0.1656 btemp is -0.7954
 Wref is 0.7000 bref is -1.0000
Epoch: 0101 cost= 0.00264
 Wtemp is 0.5047 btemp is -0.9280
 Wref is 0.7000 bref is -1.0000
Epoch: 0151 cost= 0.00083
 Wtemp is 0.6247 btemp is -0.9750
 Wref is 0.7000 bref is -1.0000
Epoch: 0201 cost= 0.00060
 Wtemp is 0.6672 btemp is -0.9916
 Wref is 0.7000 bref is -1.0000
Epoch: 0251 cost= 0.00057
 Wtemp is 0.6823 btemp is -0.9975
 Wref is 0.7000 bref is -1.0000
Epoch: 0301 cost= 0.00057
 Wtemp is 0.6876 btemp is -0.9996
 Wref is 0.7000 bref is -1.0000
Epoch: 0351 cost= 0.00056
 Wtemp is 0.6895 btemp is -1.0003
```

```
 Wref is 0.7000 bref is -1.0000
Epoch: 0401 cost= 0.00056
 Wtemp is 0.6901 btemp is -1.0006
 Wref is 0.7000 bref is -1.0000
Epoch: 0451 cost= 0.00056
 Wtemp is 0.6904 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 0501 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 0551 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 0601 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 0651 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 0701 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 0751 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 0801 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 0851 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 0901 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 0951 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1001 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1051 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1101 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1151 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1201 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1251 cost= 0.00056
```

```
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1301 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1351 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1401 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1451 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1501 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1551 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1601 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1651 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1701 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1751 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1801 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1851 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1901 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
Epoch: 1951 cost= 0.00056
 Wtemp is 0.6905 btemp is -1.0007
 Wref is 0.7000 bref is -1.0000
```

```
# Plot Results
plt.figure(2)
plt.plot(train_X[0, :], ref_Y[0, :], 'ro', label='Original data'
)
plt.plot(train_X[0, :], train_Y[0, :], 'bo', label='Training dat
a')
plt.plot(train_X[0, :], fopt[0, :], 'k-', label='Fitted Line')
plt.axis('equal')
plt.legend(loc='lower right')
```

```
<matplotlib.legend.Legend at 0x7ff520245b00>
```



LOGISTIC REGRESSION WITH MNIST

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
print ("PACKAGES LOADED")
```

```
PACKAGES LOADED
```

DOWNLOAD AND EXTRACT MNIST DATASET

```
mnist      = input_data.read_data_sets('data/', one_hot=True)
trainimg   = mnist.train.images
trainlabel = mnist.train.labels
testimg    = mnist.test.images
testlabel  = mnist.test.labels
print ("MNIST loaded")
```

```
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
MNIST loaded
```

CREATE TENSOR GRAPH FOR LOGISTIC REGRESSION

```
x = tf.placeholder("float", [None, 784])
y = tf.placeholder("float", [None, 10])  # None is for infinite
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
# LOGISTIC REGRESSION MODEL
actv = tf.nn.softmax(tf.matmul(x, W) + b)
# COST FUNCTION
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(actv), reduction_i
ndices=1))
# OPTIMIZER
learning_rate = 0.01
optm = tf.train.GradientDescentOptimizer(learning_rate).minimize
(cost)
```

PREDICTION AND ACCURACY

```
# PREDICTION
pred = tf.equal(tf.argmax(actv, 1), tf.argmax(y, 1))
# ACCURACY
accr = tf.reduce_mean(tf.cast(pred, "float"))
# INITIALIZER
init = tf.initialize_all_variables()
```

TRAIN MODEL

```
training_epochs = 50
batch_size      = 100
display_step    = 5
# SESSION
sess = tf.Session()
sess.run(init)
# MINI-BATCH LEARNING
for epoch in range(training_epochs):
    avg_cost = 0.
    num_batch = int(mnist.train.num_examples/batch_size)
    for i in range(num_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        sess.run(optm, feed_dict={x: batch_xs, y: batch_ys})
        feeds = {x: batch_xs, y: batch_ys}
        avg_cost += sess.run(cost, feed_dict=feeds)/num_batch
    # DISPLAY
    if epoch % display_step == 0:
        feeds_train = {x: batch_xs, y: batch_ys}
        feeds_test = {x: mnist.test.images, y: mnist.test.labels
}
        train_acc = sess.run(accr, feed_dict=feeds_train)
        test_acc = sess.run(accr, feed_dict=feeds_test)
        print ("Epoch: %03d/%03d cost: %.9f train_acc: %.3f test
_acc: %.3f"
               % (epoch, training_epochs, avg_cost, train_acc, t
est_acc))
print ("DONE")
```

```
Epoch: 000/050 cost: 1.176559254 train_acc: 0.870 test_acc: 0.85
2
Epoch: 005/050 cost: 0.440937506 train_acc: 0.930 test_acc: 0.89
5
Epoch: 010/050 cost: 0.383336526 train_acc: 0.900 test_acc: 0.90
4
Epoch: 015/050 cost: 0.357268913 train_acc: 0.880 test_acc: 0.90
9
Epoch: 020/050 cost: 0.341493352 train_acc: 0.970 test_acc: 0.91
2
Epoch: 025/050 cost: 0.330508839 train_acc: 0.890 test_acc: 0.91
4
Epoch: 030/050 cost: 0.322364672 train_acc: 0.880 test_acc: 0.91
6
Epoch: 035/050 cost: 0.315942195 train_acc: 0.960 test_acc: 0.91
7
Epoch: 040/050 cost: 0.310731307 train_acc: 0.910 test_acc: 0.91
8
Epoch: 045/050 cost: 0.306349064 train_acc: 0.970 test_acc: 0.91
9
DONE
```

LOGISTIC REGRESSION WITH CUSTOM DATA

```
import os
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
print ("Packages loaded")
```

```
Packages loaded
```

Load data

```
# Load them!
cwd = os.getcwd()
loadpath = cwd + "/data/custom_data.npz"
l = np.load(loadpath)

# See what's in here
print (l.files)

# Parse data
trainimg = l['trainimg']
trainlabel = l['trainlabel']
testimg = l['testimg']
testlabel = l['testlabel']
use_gray = l['use_gray']
ntrain = trainimg.shape[0]
nclass = trainlabel.shape[1]
dim    = trainimg.shape[1]
ntest  = testimg.shape[0]
print ("%d train images loaded" % (ntrain))
print ("%d test images loaded" % (ntest))
print ("%d dimensional input" % (dim))
print ("%d classes" % (nclass))
```

```
['trainlabel', 'imgsize', 'trainimg', 'testimg', 'testlabel', 'u
se_gray']
52 train images loaded
35 test images loaded
4096 dimensional input
2 classes
```

Define network

```
tf.set_random_seed(0)
# Parameters of Logistic Regression
learning_rate   = 0.001
training_epochs = 1000
batch_size      = 10
display_step    = 100

# Create Graph for Logistic Regression
x = tf.placeholder("float", [None, dim])
y = tf.placeholder("float", [None, nclass])
W = tf.Variable(tf.zeros([dim, nclass]), name = 'weights')
b = tf.Variable(tf.zeros([nclass]))
```

Define functions

```
WEIGHT_DECAY_FACTOR = 1 #  0.000001
l2_loss = tf.add_n([tf.nn.l2_loss(v)
            for v in tf.trainable_variables()])
_pred = tf.nn.softmax(tf.matmul(x, W) + b)
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(_pred)
                                     , reduction_indices=1))
cost = cost + WEIGHT_DECAY_FACTOR*l2_loss
optm = tf.train.GradientDescentOptimizer(
    learning_rate).minimize(cost)
_corr = tf.equal(tf.argmax(_pred, 1), tf.argmax(y, 1))
accr = tf.reduce_mean(tf.cast(_corr, tf.float32))
init = tf.initialize_all_variables()
print ("Functions ready")
```

```
Functions ready
```

Optimize

```
# Launch the graph
sess = tf.Session()
sess.run(init)
# Training cycle
for epoch in range(training_epochs):
    avg_cost = 0.
    num_batch = int(ntrain/batch_size)
    # Loop over all batches
    for i in range(num_batch):
        randidx = np.random.randint(ntrain, size=batch_size)
        batch_xs = trainimg[randidx, :]
        batch_ys = trainlabel[randidx, :]
        # Fit training using batch data
        sess.run(optm, feed_dict={x: batch_xs, y: batch_ys})
        # Compute average loss
        avg_cost += sess.run(cost
            , feed_dict={x: batch_xs, y: batch_ys})/num_batch

    # Display logs per epoch step
    if epoch % display_step == 0:
        print ("Epoch: %03d/%03d cost: %.9f" %
               (epoch, training_epochs, avg_cost))
        train_acc = sess.run(accr, feed_dict={x: batch_xs, y: ba
tch_ys})
        print (" Training accuracy: %.3f" % (train_acc))
        test_acc = sess.run(accr, feed_dict={x: testimg, y: test
label})
        print (" Test accuracy: %.3f" % (test_acc))
print ("Optimization Finished!")
```

```
Epoch: 000/1000 cost: 0.626751423
 Training accuracy: 0.600
 Test accuracy: 0.686
Epoch: 100/1000 cost: 0.438359487
 Training accuracy: 0.800
 Test accuracy: 0.714
Epoch: 200/1000 cost: 0.367508155
 Training accuracy: 0.900
 Test accuracy: 0.686
Epoch: 300/1000 cost: 0.363990796
 Training accuracy: 1.000
 Test accuracy: 0.714
Epoch: 400/1000 cost: 0.406193763
 Training accuracy: 0.900
 Test accuracy: 0.714
Epoch: 500/1000 cost: 0.400928861
 Training accuracy: 0.900
 Test accuracy: 0.714
Epoch: 600/1000 cost: 0.366956294
 Training accuracy: 0.900
 Test accuracy: 0.686
Epoch: 700/1000 cost: 0.334192312
 Training accuracy: 1.000
 Test accuracy: 0.686
Epoch: 800/1000 cost: 0.392353541
 Training accuracy: 1.000
 Test accuracy: 0.714
Epoch: 900/1000 cost: 0.383330226
 Training accuracy: 1.000
 Test accuracy: 0.714
Optimization Finished!
```

CLOSE SESSION

```
sess.close()
print ("Session closed.")
```

```
Session closed.
```

Multi-Layer Perceptron (MLP)

MULTI-LAYER PERCEPTRON ON MNIST

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
%matplotlib inline
print ("PACKAGES LOADED")
```

```
PACKAGES LOADED
```

LOAD MNIST

```
mnist = input_data.read_data_sets('data/', one_hot=True)
```

```
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
```

DEFINE MODEL

```
# NETWORK TOPOLOGIES
n_hidden_1 = 256
n_hidden_2 = 128
n_input    = 784
n_classes  = 10

# INPUTS AND OUTPUTS
x = tf.placeholder("float", [None, n_input])
y = tf.placeholder("float", [None, n_classes])

# NETWORK PARAMETERS
stddev = 0.1
weights = {
    'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1], st
ddev=stddev)),
    'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2],
 stddev=stddev)),
    'out': tf.Variable(tf.random_normal([n_hidden_2, n_classes],
 stddev=stddev))
}
biases = {
    'b1': tf.Variable(tf.random_normal([n_hidden_1])),
    'b2': tf.Variable(tf.random_normal([n_hidden_2])),
    'out': tf.Variable(tf.random_normal([n_classes]))
}
print ("NETWORK READY")
```

```
NETWORK READY
```

MLP AS A FUNCTION

```
def multilayer_perceptron(_X, _weights, _biases):
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(_X, _weights['h1'])
, _biases['b1']))
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, _weights['
h2']), _biases['b2']))
    return (tf.matmul(layer_2, _weights['out']) + _biases['out']
)
```

DEFINE FUNCTIONS

```
# PREDICTION
pred = multilayer_perceptron(x, weights, biases)

# LOSS AND OPTIMIZER
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pr
ed, y))
# optm = tf.train.GradientDescentOptimizer(learning_rate=0.001).
minimize(cost)
optm = tf.train.AdamOptimizer(learning_rate=0.001).minimize(cost
)
corr = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accr = tf.reduce_mean(tf.cast(corr, "float"))

# INITIALIZER
init = tf.initialize_all_variables()
print ("FUNCTIONS READY")
```

```
FUNCTIONS READY
```

RUN

```
# PARAMETERS
training_epochs = 20
batch_size      = 100
display_step    = 4
# LAUNCH THE GRAPH
sess = tf.Session()
sess.run(init)
# OPTIMIZE
for epoch in range(training_epochs):
    avg_cost = 0.
    total_batch = int(mnist.train.num_examples/batch_size)
    # ITERATION
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feeds = {x: batch_xs, y: batch_ys}
        sess.run(optm, feed_dict=feeds)
        avg_cost += sess.run(cost, feed_dict=feeds)
    avg_cost = avg_cost / total_batch
    # DISPLAY
    if (epoch+1) % display_step == 0:
        print ("Epoch: %03d/%03d cost: %.9f" % (epoch, training_
epochs, avg_cost))
        feeds = {x: batch_xs, y: batch_ys}
        train_acc = sess.run(accr, feed_dict=feeds)
        print ("TRAIN ACCURACY: %.3f" % (train_acc))
        feeds = {x: mnist.test.images, y: mnist.test.labels}
        test_acc = sess.run(accr, feed_dict=feeds)
        print ("TEST ACCURACY: %.3f" % (test_acc))
print ("OPTIMIZATION FINISHED")
```

```
Epoch: 003/020 cost: 0.124787590
TRAIN ACCURACY: 0.960
TEST ACCURACY: 0.961
Epoch: 007/020 cost: 0.050386614
TRAIN ACCURACY: 0.970
TEST ACCURACY: 0.975
Epoch: 011/020 cost: 0.019682230
TRAIN ACCURACY: 1.000
TEST ACCURACY: 0.979
Epoch: 015/020 cost: 0.007102341
TRAIN ACCURACY: 1.000
TEST ACCURACY: 0.979
Epoch: 019/020 cost: 0.002848990
TRAIN ACCURACY: 1.000
TEST ACCURACY: 0.979
OPTIMIZATION FINISHED
```

DEEPER MULTILAYER PERCEPTRON WITH DROPOUT

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
%matplotlib inline
```

LOAD MNIST

```
mnist = input_data.read_data_sets('data/', one_hot=True)
```

```
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
```

DEFINE MODEL

```
# NETWORK TOPOLOGIES
n_input    = 784 # MNIST data input (img shape: 28*28)
n_hidden_1 = 512 # 1st layer num features
n_hidden_2 = 512 # 2nd layer num features
n_hidden_3 = 256 # 3rd layer num features
n_classes  = 10 # MNIST total classes (0-9 digits)
# INPUT AND OUTPUT
x = tf.placeholder("float", [None, n_input])
y = tf.placeholder("float", [None, n_classes])
dropout_keep_prob = tf.placeholder("float")
# VARIABLES
stddev = 0.05
weights = {
    'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1], st
ddev=stddev)),
    'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2],
 stddev=stddev)),
    'h3': tf.Variable(tf.random_normal([n_hidden_2, n_hidden_3],
 stddev=stddev)),
    'out': tf.Variable(tf.random_normal([n_hidden_3, n_classes],
 stddev=stddev))
}
biases = {
    'b1': tf.Variable(tf.random_normal([n_hidden_1])),
    'b2': tf.Variable(tf.random_normal([n_hidden_2])),
    'b3': tf.Variable(tf.random_normal([n_hidden_3])),
    'out': tf.Variable(tf.random_normal([n_classes]))
}
print ("NETWORK READY")
```

```
NETWORK READY
```

```
def multilayer_perceptron(_X, _weights, _biases, _keep_prob):
    x_1 = tf.nn.relu(tf.add(tf.matmul(_X, _weights['h1']), _bias
es['b1']))
    layer_1 = x_1
    x_2 = tf.nn.relu(tf.add(tf.matmul(layer_1, _weights['h2']),
_biases['b2']))
    layer_2 = x_2
    x_3 = tf.nn.relu(tf.add(tf.matmul(layer_2, _weights['h3']),
_biases['b3']))
    layer_3 = tf.nn.dropout(x_3, _keep_prob)
    return (tf.matmul(layer_3, _weights['out']) + _biases['out']
)
```

DEFINE FUNCTIONS

```
# PREDICTION
pred = multilayer_perceptron(x, weights, biases, dropout_keep_pr
ob)

# LOSS AND OPTIMIZER
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pr
ed, y))
optm = tf.train.AdamOptimizer(learning_rate=0.001).minimize(cost
)
corr = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accr = tf.reduce_mean(tf.cast(corr, "float"))

# INITIALIZER
init = tf.initialize_all_variables()
print ("FUNCTIONS READY")
```

```
FUNCTIONS READY
```

RUN

```
# PARAMETERS
training_epochs = 20
batch_size      = 100
display_step    = 4
# LAUNCH THE GRAPH
sess = tf.Session()
sess.run(init)
# OPTIMIZE
for epoch in range(training_epochs):
    avg_cost = 0.
    total_batch = int(mnist.train.num_examples/batch_size)
    # ITERATION
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feeds = {x: batch_xs, y: batch_ys, dropout_keep_prob: 0.6
}
        sess.run(optm, feed_dict=feeds)
        feeds = {x: batch_xs, y: batch_ys, dropout_keep_prob: 1.0
}
        avg_cost += sess.run(cost, feed_dict=feeds)
    avg_cost = avg_cost / total_batch
    # DISPLAY
    if (epoch+1) % display_step == 0:
        print ("Epoch: %03d/%03d cost: %.9f" % (epoch, training_
epochs, avg_cost))
        feeds = {x: batch_xs, y: batch_ys, dropout_keep_prob: 1.0
}
        train_acc = sess.run(accr, feed_dict=feeds)
        print ("TRAIN ACCURACY: %.3f" % (train_acc))
        feeds = {x: mnist.test.images, y: mnist.test.labels, dro
pout_keep_prob: 1.0}
        test_acc = sess.run(accr, feed_dict=feeds)
        print ("TEST ACCURACY: %.3f" % (test_acc))
print ("OPTIMIZATION FINISHED")
```

```
Epoch: 003/020 cost: 0.031996857
TRAIN ACCURACY: 1.000
TEST ACCURACY: 0.975
Epoch: 007/020 cost: 0.009672010
TRAIN ACCURACY: 1.000
TEST ACCURACY: 0.977
Epoch: 011/020 cost: 0.004559030
TRAIN ACCURACY: 0.990
TEST ACCURACY: 0.979
Epoch: 015/020 cost: 0.003751091
TRAIN ACCURACY: 1.000
TEST ACCURACY: 0.980
Epoch: 019/020 cost: 0.002512690
TRAIN ACCURACY: 1.000
TEST ACCURACY: 0.982
OPTIMIZATION FINISHED
```

```
"""
 Deeper Multi-Layer Pecptron with XAVIER Init
 Xavier init from {Project: https://github.com/aymericdamien/Ten
sorFlow-Examples/}
 @Sungjoon Choi (sungjoon.choi@cpslab.snu.ac.kr)
"""
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
%matplotlib inline
```

```
mnist = input_data.read_data_sets('data/', one_hot=True)
```

```
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
```

```
# Xavier Init
def xavier_init(n_inputs, n_outputs, uniform=True):
  """Set the parameter initialization using the method described
.
  This method is designed to keep the scale of the gradients rou
ghly the same
  in all layers.
  Xavier Glorot and Yoshua Bengio (2010):
           Understanding the difficulty of training deep feedfor
ward neural
           networks. International conference on artificial inte
lligence and
           statistics.
  Args:
    n_inputs: The number of input nodes into each output.
    n_outputs: The number of output nodes for each input.
    uniform: If true use a uniform distribution, otherwise use a
 normal.
  Returns:
    An initializer.
  """
  if uniform:
    # 6 was used in the paper.
    init_range = tf.sqrt(6.0 / (n_inputs + n_outputs))
    return tf.random_uniform_initializer(-init_range, init_range
)
  else:
    # 3 gives us approximately the same limits as above since th
is repicks
    # values greater than 2 standard deviations from the mean.
    stddev = tf.sqrt(3.0 / (n_inputs + n_outputs))
    return tf.truncated_normal_initializer(stddev=stddev)
```

```
# Parameters
learning_rate   = 0.001
training_epochs = 50
batch_size      = 100
display_step    = 1

# Network Parameters
n_input    = 784 # MNIST data input (img shape: 28*28)
n_hidden_1 = 256 # 1st layer num features
n_hidden_2 = 256 # 2nd layer num features
n_hidden_3 = 256 # 3rd layer num features
n_hidden_4 = 256 # 4th layer num features
n_classes  = 10 # MNIST total classes (0-9 digits)

# tf Graph input
x = tf.placeholder("float", [None, n_input])
y = tf.placeholder("float", [None, n_classes])
```

```
dropout_keep_prob = tf.placeholder("float")

# Create model
def multilayer_perceptron(_X, _weights, _biases, _keep_prob):
    layer_1 = tf.nn.dropout(tf.nn.relu(tf.add(tf.matmul(_X, _wei
ghts['h1']), _biases['b1'])), _keep_prob)
    layer_2 = tf.nn.dropout(tf.nn.relu(tf.add(tf.matmul(layer_1,
 _weights['h2']), _biases['b2'])), _keep_prob)
    layer_3 = tf.nn.dropout(tf.nn.relu(tf.add(tf.matmul(layer_2,
 _weights['h3']), _biases['b3'])), _keep_prob)
    layer_4 = tf.nn.dropout(tf.nn.relu(tf.add(tf.matmul(layer_3,
 _weights['h4']), _biases['b4'])), _keep_prob)
    return (tf.matmul(layer_4, _weights['out']) + _biases['out']
) # No need to use softmax??

# Store layers weight & bias
weights = {
    'h1': tf.get_variable("h1", shape=[n_input, n_hidden_1],
initializer=xavier_init(n_input,n_hidden_1)),
    'h2': tf.get_variable("h2", shape=[n_hidden_1, n_hidden_2],
initializer=xavier_init(n_hidden_1,n_hidden_2)),
    'h3': tf.get_variable("h3", shape=[n_hidden_2, n_hidden_3],
initializer=xavier_init(n_hidden_2,n_hidden_3)),
    'h4': tf.get_variable("h4", shape=[n_hidden_3, n_hidden_4],
initializer=xavier_init(n_hidden_3,n_hidden_4)),
    'out': tf.get_variable("out", shape=[n_hidden_4, n_classes],
 initializer=xavier_init(n_hidden_4,n_classes))
}
biases = {
    'b1': tf.Variable(tf.random_normal([n_hidden_1])),
    'b2': tf.Variable(tf.random_normal([n_hidden_2])),
    'b3': tf.Variable(tf.random_normal([n_hidden_3])),
    'b4': tf.Variable(tf.random_normal([n_hidden_4])),
    'out': tf.Variable(tf.random_normal([n_classes]))
}

# Construct model
pred = multilayer_perceptron(x, weights, biases, dropout_keep_pr
ob)

# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pr
ed, y)) # Softmax loss
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).
minimize(cost) # Adam Optimizer
# optimizer = tf.train.MomentumOptimizer(learning_rate=learning_
rate, momentum=0.8).minimize(cost) # Adam Optimizer

# Accuracy
correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1
))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

```
# Initializing the variables
init = tf.initialize_all_variables()

print ("Network Ready")
```

```
Network Ready
```

```
# Launch the graph
sess = tf.Session()
sess.run(init)

# Training cycle
for epoch in range(training_epochs):
    avg_cost = 0.
    total_batch = int(mnist.train.num_examples/batch_size)
    # Loop over all batches
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        # Fit training using batch data
        sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys,
 dropout_keep_prob: 0.7})
        # Compute average loss
        avg_cost += sess.run(cost, feed_dict={x: batch_xs, y: ba
tch_ys, dropout_keep_prob:1.})/total_batch
    # Display logs per epoch step
    if epoch % display_step == 0:
        print ("Epoch: %03d/%03d cost: %.9f" % (epoch, training_
epochs, avg_cost))
        train_acc = sess.run(accuracy, feed_dict={x: batch_xs, y
: batch_ys, dropout_keep_prob:1.})
        print ("Training accuracy: %.3f" % (train_acc))

print ("Optimization Finished!")
```

```
Epoch: 000/050 cost: 0.418188300
Training accuracy: 0.970
Epoch: 001/050 cost: 0.135868739
Training accuracy: 0.960
Epoch: 002/050 cost: 0.095398066
Training accuracy: 0.980
Epoch: 003/050 cost: 0.075697627
Training accuracy: 0.970
Epoch: 004/050 cost: 0.061306230
Training accuracy: 0.980
Epoch: 005/050 cost: 0.052467199
Training accuracy: 0.970
Epoch: 006/050 cost: 0.046108751
Training accuracy: 0.970
```

```
Epoch: 007/050 cost: 0.040648738
Training accuracy: 1.000
Epoch: 008/050 cost: 0.034662794
Training accuracy: 0.980
Epoch: 009/050 cost: 0.030895058
Training accuracy: 0.980
Epoch: 010/050 cost: 0.027404196
Training accuracy: 0.990
Epoch: 011/050 cost: 0.025599543
Training accuracy: 0.990
Epoch: 012/050 cost: 0.022524802
Training accuracy: 0.990
Epoch: 013/050 cost: 0.020978481
Training accuracy: 1.000
Epoch: 014/050 cost: 0.018992848
Training accuracy: 1.000
Epoch: 015/050 cost: 0.018180760
Training accuracy: 1.000
Epoch: 016/050 cost: 0.015807947
Training accuracy: 1.000
Epoch: 017/050 cost: 0.015066529
Training accuracy: 1.000
Epoch: 018/050 cost: 0.013667117
Training accuracy: 0.990
Epoch: 019/050 cost: 0.012571550
Training accuracy: 1.000
Epoch: 020/050 cost: 0.011941066
Training accuracy: 0.990
Epoch: 021/050 cost: 0.011349857
Training accuracy: 0.990
Epoch: 022/050 cost: 0.010185919
Training accuracy: 1.000
Epoch: 023/050 cost: 0.010433348
Training accuracy: 1.000
Epoch: 024/050 cost: 0.009212835
Training accuracy: 1.000
Epoch: 025/050 cost: 0.008935386
Training accuracy: 0.990
Epoch: 026/050 cost: 0.007533159
Training accuracy: 1.000
Epoch: 027/050 cost: 0.008139531
Training accuracy: 1.000
Epoch: 028/050 cost: 0.007539478
Training accuracy: 1.000
Epoch: 029/050 cost: 0.007227370
Training accuracy: 1.000
Epoch: 030/050 cost: 0.007473362
Training accuracy: 1.000
Epoch: 031/050 cost: 0.006676663
Training accuracy: 0.980
Epoch: 032/050 cost: 0.005731412
Training accuracy: 1.000
Epoch: 033/050 cost: 0.005602606
```

```
Training accuracy: 0.990
Epoch: 034/050 cost: 0.005471543
Training accuracy: 1.000
Epoch: 035/050 cost: 0.005467167
Training accuracy: 1.000
Epoch: 036/050 cost: 0.006077243
Training accuracy: 1.000
Epoch: 037/050 cost: 0.005801255
Training accuracy: 1.000
Epoch: 038/050 cost: 0.005572326
Training accuracy: 1.000
Epoch: 039/050 cost: 0.005355799
Training accuracy: 1.000
Epoch: 040/050 cost: 0.004890651
Training accuracy: 1.000
Epoch: 041/050 cost: 0.004345889
Training accuracy: 1.000
Epoch: 042/050 cost: 0.004596357
Training accuracy: 1.000
Epoch: 043/050 cost: 0.003729049
Training accuracy: 1.000
Epoch: 044/050 cost: 0.004191519
Training accuracy: 1.000
Epoch: 045/050 cost: 0.004694648
Training accuracy: 1.000
Epoch: 046/050 cost: 0.003776975
Training accuracy: 1.000
Epoch: 047/050 cost: 0.004078514
Training accuracy: 0.990
Epoch: 048/050 cost: 0.003377332
Training accuracy: 1.000
Epoch: 049/050 cost: 0.003732143
Training accuracy: 1.000
Optimization Finished!
```

```
test_acc = sess.run(accuracy, feed_dict={x: mnist.test.images, y
: mnist.test.labels, dropout_keep_prob:1.})
print ("Training accuracy: %.3f" % (test_acc))
```

```
Training accuracy: 0.982
```

MULTI-LAYER PERCEPTRON WITH CUSTOM DATA

```
import os
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
%matplotlib inline
print ("Packages loaded")
```

```
Packages loaded
```

LOAD DATA

```
# Load them!
cwd = os.getcwd()
loadpath = cwd + "/data/custom_data.npz"
l = np.load(loadpath)

# See what's in here
print (l.files)

# Parse data
trainimg = l['trainimg']
trainlabel = l['trainlabel']
testimg = l['testimg']
testlabel = l['testlabel']
imgsize = l['imgsize']
ntrain = trainimg.shape[0]
nclass = trainlabel.shape[1]
dim    = trainimg.shape[1]
ntest  = testimg.shape[0]
print ("%d train images loaded" % (ntrain))
print ("%d test images loaded" % (ntest))
print ("%d dimensional input" % (dim))
print ("Image size is %s" % (imgsize))
print ("%d classes" % (nclass))
```

```
['trainlabel', 'imgsize', 'trainimg', 'testimg', 'testlabel', 'u
se_gray']
408 train images loaded
273 test images loaded
4096 dimensional input
Image size is [64 64]
4 classes
```

DEFINE NETWORK

```
tf.set_random_seed(0)
# Parameters
learning_rate   = 0.001
training_epochs = 200
batch_size      = ntrain
display_step    = 20

# Network Parameters
n_hidden_1 = 128 # 1st layer num features
n_hidden_2 = 128 # 2nd layer num features
n_input    = dim # data input
n_classes  = nclass # total classes (0-9 digits)

# tf Graph input
x = tf.placeholder("float", [None, n_input])
y = tf.placeholder("float", [None, n_classes])

# Create model
def multilayer_perceptron(_X, _weights, _biases):
    layer_1 = tf.nn.relu(tf.add(tf.matmul(_X, _weights['h1']), _
biases['b1']))
    layer_2 = tf.nn.relu(tf.add(tf.matmul(layer_1, _weights['h2'
]), _biases['b2']))
    return tf.matmul(layer_2, _weights['out']) + _biases['out']

# Store layers weight & bias
stddev = 0.1 # <== This greatly affects accuracy!!
weights = {
    'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1], st
ddev=stddev)),
    'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2],
 stddev=stddev)),
    'out': tf.Variable(tf.random_normal([n_hidden_2, n_classes],
 stddev=stddev))
}
biases = {
    'b1': tf.Variable(tf.random_normal([n_hidden_1])),
    'b2': tf.Variable(tf.random_normal([n_hidden_2])),
    'out': tf.Variable(tf.random_normal([n_classes]))
}
print ("Network Ready to Go!")
```

```
Network Ready to Go!
```

DEFINE FUNCTIONS

```
# Construct model
pred = multilayer_perceptron(x, weights, biases)

# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pr
ed, y))
optm = tf.train.AdamOptimizer(learning_rate=learning_rate).minim
ize(cost)
corr = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accr = tf.reduce_mean(tf.cast(corr, "float"))

# Initializing the variables
init = tf.initialize_all_variables()
print ("Functions ready")
```

```
Functions ready
```

OPTIMIZE

```
# Launch the graph
sess = tf.Session()
sess.run(init)

# Training cycle
for epoch in range(training_epochs):
    avg_cost = 0.
    total_batch = int(ntrain/batch_size)
    # Loop over all batches
    for i in range(total_batch):
        randidx = np.random.randint(ntrain, size=batch_size)
        batch_xs = trainimg[randidx, :]
        batch_ys = trainlabel[randidx, :]
        # Fit training using batch data
        sess.run(optm, feed_dict={x: batch_xs, y: batch_ys})
        # Compute average loss
        avg_cost += sess.run(cost,
                feed_dict={x: batch_xs, y: batch_ys})/total_batc
h
        # Display logs per epoch step
    if epoch % display_step == 0:
        print ("Epoch: %03d/%03d cost: %.9f" %
               (epoch, training_epochs, avg_cost))
        train_acc = sess.run(accr, feed_dict={x: batch_xs, y: ba
tch_ys})
        print (" Training accuracy: %.3f" % (train_acc))
        test_acc = sess.run(accr, feed_dict={x: testimg, y: test
label})
        print (" Test accuracy: %.3f" % (test_acc))

print ("Optimization Finished!")
```

```
Epoch: 000/200 cost: 1.271932960
 Training accuracy: 0.723
 Test accuracy: 0.707
Epoch: 020/200 cost: 0.485024929
 Training accuracy: 0.831
 Test accuracy: 0.817
Epoch: 040/200 cost: 0.269245446
 Training accuracy: 0.926
 Test accuracy: 0.857
Epoch: 060/200 cost: 0.158787951
 Training accuracy: 0.973
 Test accuracy: 0.883
Epoch: 080/200 cost: 0.054250188
 Training accuracy: 0.993
 Test accuracy: 0.905
Epoch: 100/200 cost: 0.034250565
 Training accuracy: 0.998
 Test accuracy: 0.908
Epoch: 120/200 cost: 0.016527731
 Training accuracy: 1.000
 Test accuracy: 0.905
Epoch: 140/200 cost: 0.013574737
 Training accuracy: 1.000
 Test accuracy: 0.894
Epoch: 160/200 cost: 0.006475344
 Training accuracy: 1.000
 Test accuracy: 0.894
Epoch: 180/200 cost: 0.004590446
 Training accuracy: 1.000
 Test accuracy: 0.894
Optimization Finished!
```

CLOSE SESSION

```
sess.close()
print ("Session closed.")
```

```
Session closed.
```

Convolutional Neural Network (CNN)

SIMPLE CONVOLUTIONAL NEURAL NETWORK

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
%matplotlib inline
print ("PACKAGES LOADED")
```

```
PACKAGES LOADED
```

LOAD MNIST

```
mnist = input_data.read_data_sets('data/', one_hot=True)
trainimg   = mnist.train.images
trainlabel = mnist.train.labels
testimg    = mnist.test.images
testlabel  = mnist.test.labels
print ("MNIST ready")
```

```
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
MNIST ready
```

SELECT DEVICE TO BE USED

```
device_type = "/gpu:1"
```

DEFINE CNN

```
with tf.device(device_type): # <= This is optional
    n_input  = 784
    n_output = 10
    weights  = {
        'wc1': tf.Variable(tf.random_normal([3, 3, 1, 64], stdde
v=0.1)),
        'wd1': tf.Variable(tf.random_normal([14*14*64, n_output]
, stddev=0.1))
    }
    biases   = {
        'bc1': tf.Variable(tf.random_normal([64], stddev=0.1)),
        'bd1': tf.Variable(tf.random_normal([n_output], stddev=0
.1))
    }
    def conv_simple(_input, _w, _b):
        # Reshape input
        _input_r = tf.reshape(_input, shape=[-1, 28, 28, 1])
        # Convolution
        _conv1 = tf.nn.conv2d(_input_r, _w['wc1'], strides=[1, 1
, 1, 1], padding='SAME')
        # Add-bias
        _conv2 = tf.nn.bias_add(_conv1, _b['bc1'])
        # Pass ReLu
        _conv3 = tf.nn.relu(_conv2)
        # Max-pooling
        _pool  = tf.nn.max_pool(_conv3, ksize=[1, 2, 2, 1], stri
des=[1, 2, 2, 1], padding='SAME')
        # Vectorize
        _dense = tf.reshape(_pool, [-1, _w['wd1'].get_shape().as
_list()[0]])
        # Fully-connected layer
        _out = tf.add(tf.matmul(_dense, _w['wd1']), _b['bd1'])
        # Return everything
        out = {
            'input_r': _input_r, 'conv1': _conv1, 'conv2': _conv
2, 'conv3': _conv3
            , 'pool': _pool, 'dense': _dense, 'out': _out
        }
        return out
print ("CNN ready")
```

```
CNN ready
```

DEFINE COMPUTATIONAL GRAPH

```
# tf Graph input
x = tf.placeholder(tf.float32, [None, n_input])
y = tf.placeholder(tf.float32, [None, n_output])
# Parameters
learning_rate   = 0.001
training_epochs = 10
batch_size      = 100
display_step    = 1
# Functions!
with tf.device(device_type): # <= This is optional
    _pred = conv_simple(x, weights, biases)['out']
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logit
s(_pred, y))
    optm = tf.train.AdamOptimizer(learning_rate=learning_rate).m
inimize(cost)
    _corr = tf.equal(tf.argmax(_pred,1), tf.argmax(y,1)) # Count
 corrects
    accr = tf.reduce_mean(tf.cast(_corr, tf.float32)) # Accuracy
    init = tf.initialize_all_variables()
# Saver
save_step = 1;
savedir = "nets/"
saver = tf.train.Saver(max_to_keep=3)
print ("Network Ready to Go!")
```

```
Network Ready to Go!
```

OPTIMIZE

DO TRAIN OR NOT

```
do_train = 1
sess = tf.Session(config=tf.ConfigProto(allow_soft_placement=True
))
sess.run(init)
```

```
if do_train == 1:
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_si
ze)
            # Fit training using batch data
            sess.run(optm, feed_dict={x: batch_xs, y: batch_ys})
            # Compute average loss
            avg_cost += sess.run(cost, feed_dict={x: batch_xs, y
: batch_ys})/total_batch

        # Display logs per epoch step
        if epoch % display_step == 0:
            print ("Epoch: %03d/%03d cost: %.9f" % (epoch, train
ing_epochs, avg_cost))
            train_acc = sess.run(accr, feed_dict={x: batch_xs, y
: batch_ys})
            print (" Training accuracy: %.3f" % (train_acc))
            test_acc = sess.run(accr, feed_dict={x: testimg, y:
testlabel})
            print (" Test accuracy: %.3f" % (test_acc))

        # Save Net
        if epoch % save_step == 0:
            saver.save(sess, "nets/cnn_mnist_simple.ckpt-" + str
(epoch))
    print ("Optimization Finished.")
```

```
Epoch: 000/010 cost: 0.326192696
 Training accuracy: 0.980
 Test accuracy: 0.959
Epoch: 001/010 cost: 0.105607550
 Training accuracy: 0.960
 Test accuracy: 0.975
Epoch: 002/010 cost: 0.072013733
 Training accuracy: 0.960
 Test accuracy: 0.979
Epoch: 003/010 cost: 0.056868095
 Training accuracy: 0.990
 Test accuracy: 0.980
Epoch: 004/010 cost: 0.047069814
 Training accuracy: 0.990
 Test accuracy: 0.983
Epoch: 005/010 cost: 0.040124569
 Training accuracy: 0.980
 Test accuracy: 0.983
Epoch: 006/010 cost: 0.035343169
 Training accuracy: 0.990
 Test accuracy: 0.983
Epoch: 007/010 cost: 0.030736405
 Training accuracy: 1.000
 Test accuracy: 0.984
Epoch: 008/010 cost: 0.026192359
 Training accuracy: 0.990
 Test accuracy: 0.983
Epoch: 009/010 cost: 0.024165640
 Training accuracy: 1.000
 Test accuracy: 0.983
Optimization Finished.
```

RESTORE

```
if do_train == 0:
    epoch = training_epochs-1
    saver.restore(sess, "nets/cnn_mnist_simple.ckpt-" + str(epoc
h))
    print ("NETWORK RESTORED")
```

LET'S SEE HOW CNN WORKS

```
with tf.device(device_type):
    conv_out = conv_simple(x, weights, biases)

input_r = sess.run(conv_out['input_r'], feed_dict={x: trainimg[0:
1, :]})
conv1   = sess.run(conv_out['conv1'], feed_dict={x: trainimg[0:1
, :]})
conv2   = sess.run(conv_out['conv2'], feed_dict={x: trainimg[0:1
, :]})
conv3   = sess.run(conv_out['conv3'], feed_dict={x: trainimg[0:1
, :]})
pool    = sess.run(conv_out['pool'], feed_dict={x: trainimg[0:1,
 :]})
dense   = sess.run(conv_out['dense'], feed_dict={x: trainimg[0:1
, :]})
out     = sess.run(conv_out['out'], feed_dict={x: trainimg[0:1,
:]})
```

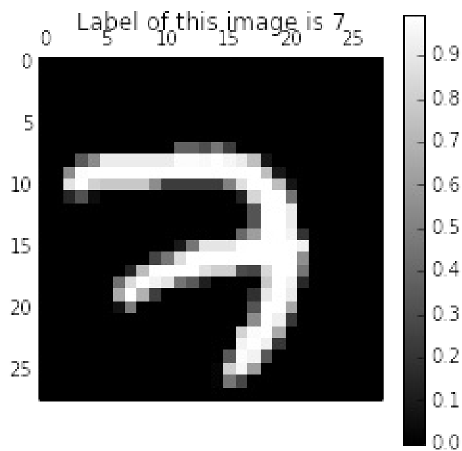
Input

```
# Let's see 'input_r'
print ("Size of 'input_r' is %s" % (input_r.shape,))
label = np.argmax(trainlabel[0, :])
print ("Label is %d" % (label))

# Plot !
plt.matshow(input_r[0, :, :, 0], cmap=plt.get_cmap('gray'))
plt.title("Label of this image is " + str(label) + "")
plt.colorbar()
plt.show()
```

```
Size of 'input_r' is (1, 28, 28, 1)
Label is 7


/usr/lib/pymodules/python2.7/matplotlib/collections.py:548: Futu
reWarning: elementwise comparison failed; returning scalar inste
ad, but in the future will perform elementwise comparison
  if self._edgecolors == 'face':
```

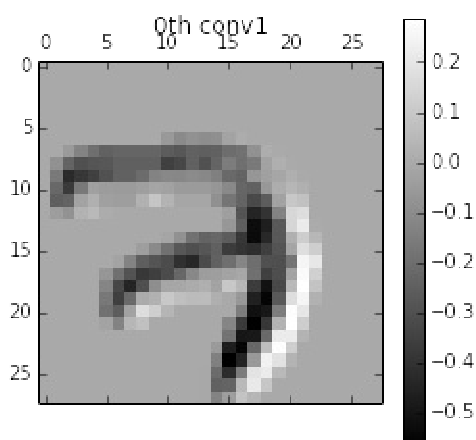


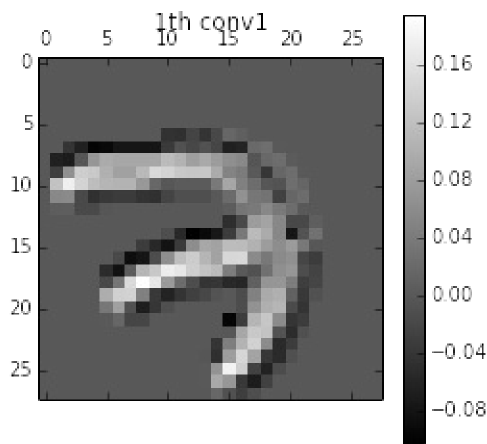
Conv1 (convolution)

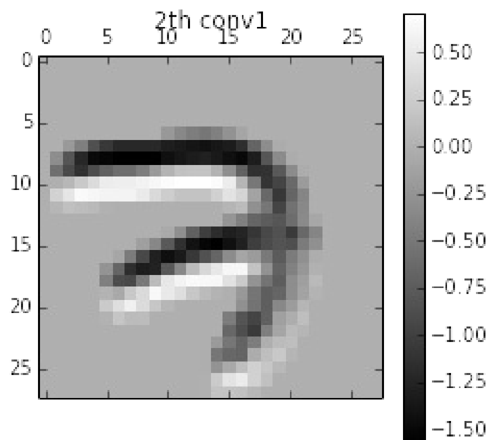
```
# Let's see 'conv1'
print ("Size of 'conv1' is %s" % (conv1.shape,))

# Plot !
for i in range(3):
    plt.matshow(conv1[0, :, :, i], cmap=plt.get_cmap('gray'))
    plt.title(str(i) + "th conv1")
    plt.colorbar()
    plt.show()
```

```
Size of 'conv1' is (1, 28, 28, 64)
```





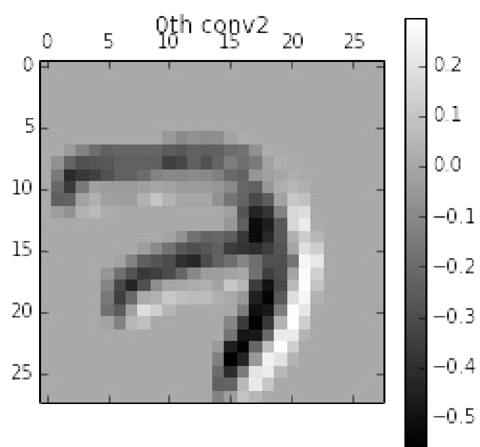


Conv2 (+bias)

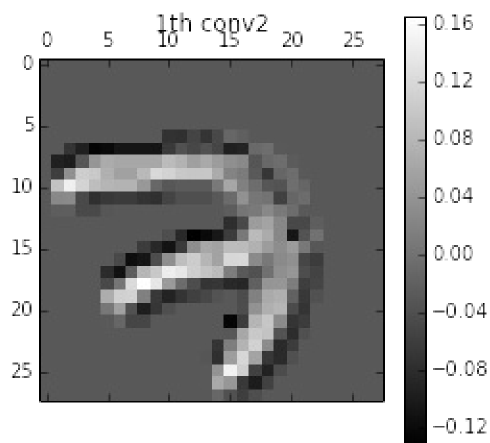
```
# Let's see 'conv2'
print ("Size of 'conv2' is %s" % (conv2.shape,))

# Plot !
for i in range(3):
    plt.matshow(conv2[0, :, :, i], cmap=plt.get_cmap('gray'))
    plt.title(str(i) + "th conv2")
    plt.colorbar()
    plt.show()
```

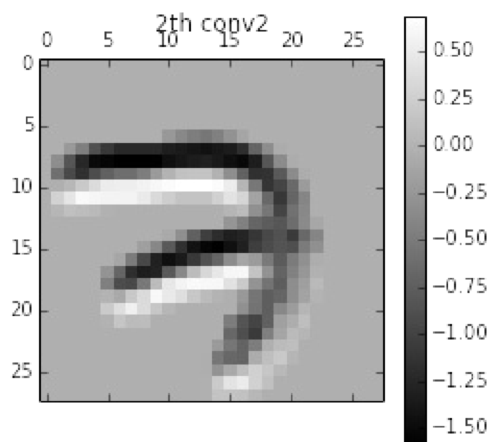
```
Size of 'conv2' is (1, 28, 28, 64)
```



0th conv2
0
5
10
15
20
25
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5



1th conv2
0
5
10
15
20
25
0.16
0.12
0.08
0.04
0.00
−0.04
−0.08
−0.12



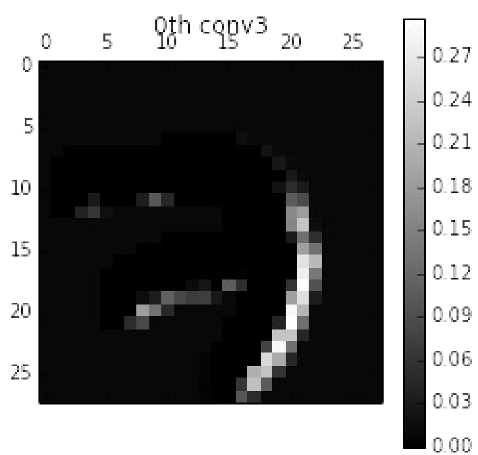
2th conv2
0
5
10
15
20
25
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50

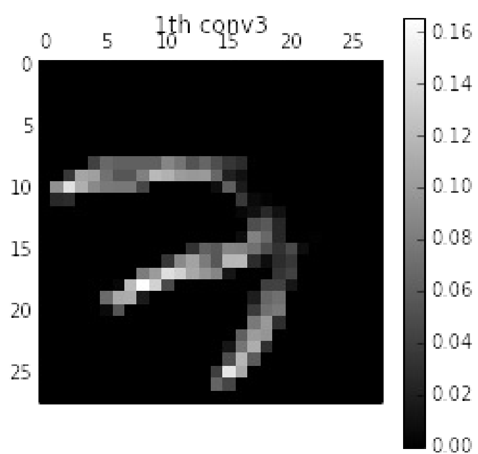
Conv3 (ReLU)

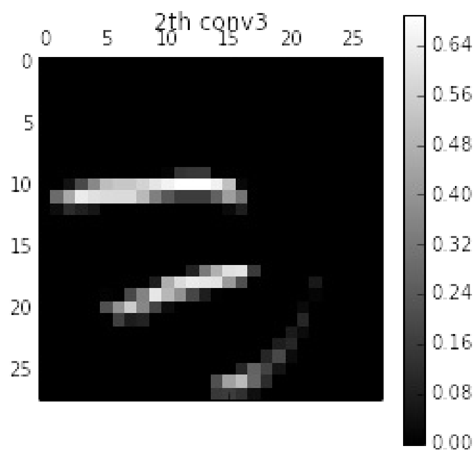
```
# Let's see 'conv3'
print ("Size of 'conv3' is %s" % (conv3.shape,))

# Plot !
for i in range(3):
    plt.matshow(conv3[0, :, :, i], cmap=plt.get_cmap('gray'))
    plt.title(str(i) + "th conv3")
    plt.colorbar()
    plt.show()
```

```
Size of 'conv3' is (1, 28, 28, 64)
```





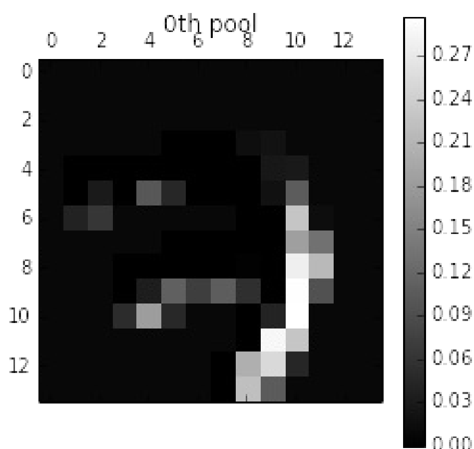


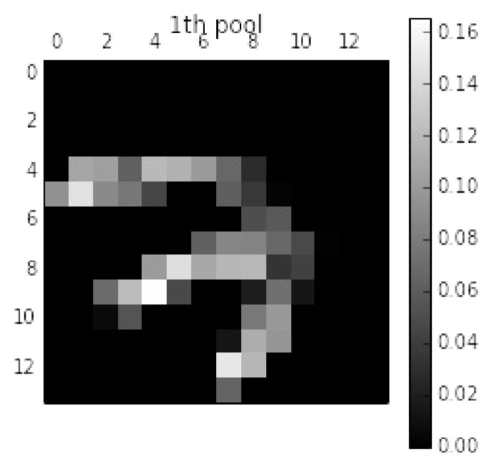
Pool (max_pool)

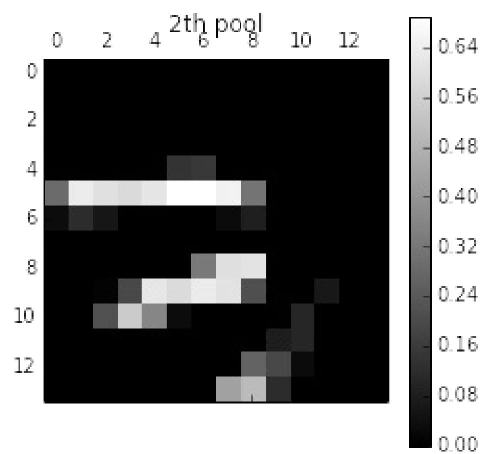
```
# Let's see 'pool'
print ("Size of 'pool' is %s" % (pool.shape,))

# Plot !
for i in range(3):
    plt.matshow(pool[0, :, :, i], cmap=plt.get_cmap('gray'))
    plt.title(str(i) + "th pool")
    plt.colorbar()
    plt.show()
```

```
Size of 'pool' is (1, 14, 14, 64)
```







Dense

```
# Let's see 'dense'
print ("Size of 'dense' is %s" % (dense.shape,))
# Let's see 'out'
print ("Size of 'out' is %s" % (out.shape,))
```

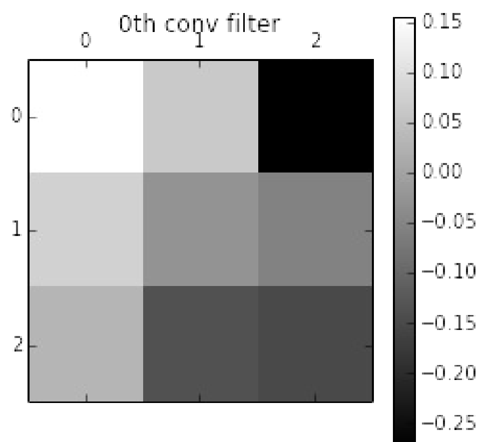
```
Size of 'dense' is (1, 12544)
Size of 'out' is (1, 10)
```

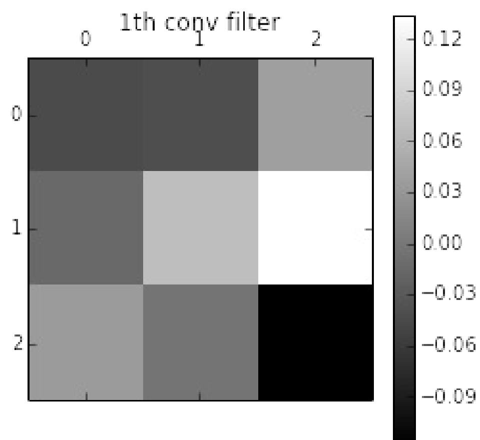
Convolution filters

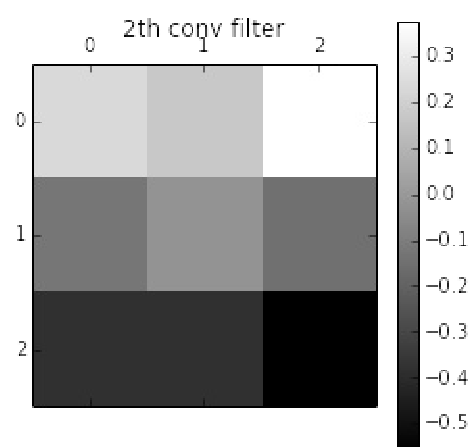
```
# Let's see weight!
wc1 = sess.run(weights['wc1'])
print ("Size of 'wc1' is %s" % (wc1.shape,))

# Plot !
for i in range(3):
    plt.matshow(wc1[:, :, 0, i], cmap=plt.get_cmap('gray'))
    plt.title(str(i) + "th conv filter")
    plt.colorbar()
    plt.show()
```

```
Size of 'wc1' is (3, 3, 1, 64)
```






2th conv filter
0
1
2
0
1
2
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5

MODERN CONVOLUTIONAL NEURAL NETWORK

DROPOUT + BATCH NORMALIZATION

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
%matplotlib inline
print ("PACKAGES LOADED")
```

```
PACKAGES LOADED
```

LOAD MNIST

```
mnist = input_data.read_data_sets('data/', one_hot=True)
trainimg   = mnist.train.images
trainlabel = mnist.train.labels
testimg    = mnist.test.images
testlabel  = mnist.test.labels
```

```
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
```

SELECT DEVICE TO BE USED

```
device_type = "/gpu:1"
```

DEFINE CNN

```
n_input  = 784
```

```
n_output = 10
with tf.device(device_type):
    weights  = {
        'wc1': tf.Variable(tf.truncated_normal([3, 3, 1, 64], st
ddev=0.1)),
        'wc2': tf.Variable(tf.truncated_normal([3, 3, 64, 128],
stddev=0.1)),
        'wd1': tf.Variable(tf.truncated_normal([7*7*128, 1024],
stddev=0.1)),
        'wd2': tf.Variable(tf.truncated_normal([1024, n_output],
 stddev=0.1))
    }
    biases   = {
        'bc1': tf.Variable(tf.random_normal([64], stddev=0.1)),
        'bc2': tf.Variable(tf.random_normal([128], stddev=0.1)),
        'bd1': tf.Variable(tf.random_normal([1024], stddev=0.1))
,
        'bd2': tf.Variable(tf.random_normal([n_output], stddev=0
.1))
    }
    def conv_basic(_input, _w, _b, _keepratio):
        # INPUT
        _input_r = tf.reshape(_input, shape=[-1, 28, 28, 1])
        # CONV LAYER 1
        _conv1 = tf.nn.conv2d(_input_r, _w['wc1'], strides=[1, 1
, 1, 1], padding='SAME')
        _mean, _var = tf.nn.moments(_conv1, [0, 1, 2])
        _conv1 = tf.nn.batch_normalization(_conv1, _mean, _var, 0
, 1, 0.0001)
        _conv1 = tf.nn.relu(tf.nn.bias_add(_conv1, _b['bc1']))
        _pool1 = tf.nn.max_pool(_conv1, ksize=[1, 2, 2, 1], stri
des=[1, 2, 2, 1], padding='SAME')
        _pool_dr1 = tf.nn.dropout(_pool1, _keepratio)
        # CONV LAYER 2
        _conv2 = tf.nn.conv2d(_pool_dr1, _w['wc2'], strides=[1, 1
, 1, 1], padding='SAME')
        _mean, _var = tf.nn.moments(_conv2, [0, 1, 2])
        _conv2 = tf.nn.batch_normalization(_conv2, _mean, _var, 0
, 1, 0.0001)
        _conv2 = tf.nn.relu(tf.nn.bias_add(_conv2, _b['bc2']))
        _pool2 = tf.nn.max_pool(_conv2, ksize=[1, 2, 2, 1], stri
des=[1, 2, 2, 1], padding='SAME')
        _pool_dr2 = tf.nn.dropout(_pool2, _keepratio)
        # VECTORIZE
        _dense1 = tf.reshape(_pool_dr2, [-1, _w['wd1'].get_shape
().as_list()[0]])
        # FULLY CONNECTED LAYER 1
        _fc1 = tf.nn.relu(tf.add(tf.matmul(_dense1, _w['wd1']),
_b['bd1']))
        _fc_dr1 = tf.nn.dropout(_fc1, _keepratio)
        # FULLY CONNECTED LAYER 2
        _out = tf.add(tf.matmul(_fc_dr1, _w['wd2']), _b['bd2'])
        # RETURN
```

```
        out = { 'input_r': _input_r, 'conv1': _conv1, 'pool1': _
pool1, 'pool1_dr1': _pool_dr1,
            'conv2': _conv2, 'pool2': _pool2, 'pool_dr2': _pool_
dr2, 'dense1': _dense1,
            'fc1': _fc1, 'fc_dr1': _fc_dr1, 'out': _out
        }
        return out
print ("CNN READY")
```

```
CNN READY
```

DEFINE COMPUTATIONAL GRAPH

```
# PLACEHOLDERS
x = tf.placeholder(tf.float32, [None, n_input])
y = tf.placeholder(tf.float32, [None, n_output])
keepratio = tf.placeholder(tf.float32)

# FUNCTIONS
with tf.device(device_type):
    _pred = conv_basic(x, weights, biases, keepratio)['out']
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logit
s(_pred, y))
    optm = tf.train.AdamOptimizer(learning_rate=0.001).minimize(
cost)
    _corr = tf.equal(tf.argmax(_pred,1), tf.argmax(y,1))
    accr = tf.reduce_mean(tf.cast(_corr, tf.float32))
    init = tf.initialize_all_variables()

# SAVER
save_step = 1
saver = tf.train.Saver(max_to_keep=3)
print ("GRAPH READY")
```

```
GRAPH READY
```

OPTIMIZE

DO TRAIN OR NOT

```
do_train = 1
sess = tf.Session(config=tf.ConfigProto(allow_soft_placement=True
))
sess.run(init)
```

```
training_epochs = 15
batch_size      = 100
display_step    = 1
if do_train == 1:
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_si
ze)
            # Fit training using batch data
            sess.run(optm, feed_dict={x: batch_xs, y: batch_ys,
keepratio:0.7})
            # Compute average loss
            avg_cost += sess.run(cost, feed_dict={x: batch_xs, y
: batch_ys, keepratio:1.})/total_batch

        # Display logs per epoch step
        if epoch % display_step == 0:
            print ("Epoch: %03d/%03d cost: %.9f" % (epoch, train
ing_epochs, avg_cost))
            train_acc = sess.run(accr, feed_dict={x: batch_xs, y
: batch_ys, keepratio:1.})
            print (" Training accuracy: %.3f" % (train_acc))
            test_acc = sess.run(accr, feed_dict={x: testimg, y:
testlabel, keepratio:1.})
            print (" Test accuracy: %.3f" % (test_acc))

        # Save Net
        if epoch % save_step == 0:
            saver.save(sess, "nets/cnn_mnist_basic.ckpt-" + str(
epoch))

    print ("OPTIMIZATION FINISHED")
```

```
Epoch: 000/015 cost: 0.707494674
 Training accuracy: 0.990
 Test accuracy: 0.962
Epoch: 001/015 cost: 0.091100394
 Training accuracy: 0.970
 Test accuracy: 0.976
Epoch: 002/015 cost: 0.062834177
 Training accuracy: 0.970
 Test accuracy: 0.982
Epoch: 003/015 cost: 0.050124394
 Training accuracy: 0.980
 Test accuracy: 0.986
Epoch: 004/015 cost: 0.041069326
 Training accuracy: 0.990
 Test accuracy: 0.985
Epoch: 005/015 cost: 0.035412403
 Training accuracy: 1.000
 Test accuracy: 0.989
Epoch: 006/015 cost: 0.030160291
 Training accuracy: 0.990
 Test accuracy: 0.991
Epoch: 007/015 cost: 0.026010393
 Training accuracy: 1.000
 Test accuracy: 0.990
Epoch: 008/015 cost: 0.024443544
 Training accuracy: 0.980
 Test accuracy: 0.986
Epoch: 009/015 cost: 0.021579011
 Training accuracy: 0.980
 Test accuracy: 0.990
Epoch: 010/015 cost: 0.018166464
 Training accuracy: 0.990
 Test accuracy: 0.990
Epoch: 011/015 cost: 0.017664607
 Training accuracy: 1.000
 Test accuracy: 0.991
Epoch: 012/015 cost: 0.015602452
 Training accuracy: 0.990
 Test accuracy: 0.990
Epoch: 013/015 cost: 0.013644544
 Training accuracy: 1.000
 Test accuracy: 0.991
Epoch: 014/015 cost: 0.011789304
 Training accuracy: 1.000
 Test accuracy: 0.993
OPTIMIZATION FINISHED
```

RESTORE

```
if do_train == 0:
    epoch = training_epochs-1
    saver.restore(sess, "nets/cnn_mnist_basic.ckpt-" + str(epoch
))
```

COMPUTE TEST ACCURACY

```
test_acc = sess.run(accr, feed_dict={x: testimg, y: testlabel, k
eepratio:1.})
print (" TEST ACCURACY: %.3f" % (test_acc))
```

```
 TEST ACCURACY: 0.993
```

CONVOLUTIONAL NEURAL NETWORK WITH CUSTOM DATA

```
import os
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
%matplotlib inline
print ("Packages loaded")
```

```
Packages loaded
```

LOAD DATA

```
# Load them!
cwd = os.getcwd()
loadpath = cwd + "/data/custom_data.npz"
l = np.load(loadpath)

# See what's in here
print (l.files)

# Parse data
trainimg = l['trainimg']
trainlabel = l['trainlabel']
testimg = l['testimg']
testlabel = l['testlabel']
imgsize = l['imgsize']
use_gray = l['use_gray']
ntrain = trainimg.shape[0]
nclass = trainlabel.shape[1]
dim    = trainimg.shape[1]
ntest  = testimg.shape[0]
print ("%d train images loaded" % (ntrain))
print ("%d test images loaded" % (ntest))
print ("%d dimensional input" % (dim))
print ("Image size is %s" % (imgsize))
print ("%d classes" % (nclass))
```

```
['trainlabel', 'imgsize', 'trainimg', 'testimg', 'testlabel', 'u
se_gray']
408 train images loaded
273 test images loaded
4096 dimensional input
Image size is [64 64]
4 classes
```

DEFINE NETWORK

```
tf.set_random_seed(0)
n_input  = dim
n_output = nclass
if use_gray:
    weights  = {
        'wc1': tf.Variable(tf.random_normal([5, 5, 1, 128], stdd
ev=0.1)),
        'wc2': tf.Variable(tf.random_normal([5, 5, 128, 128], st
ddev=0.1)),
        'wd1': tf.Variable(tf.random_normal(
                [(int)(imgsize[0]/4*imgsize[1]/4)*128, 128], std
dev=0.1)),
        'wd2': tf.Variable(tf.random_normal([128, n_output], std
dev=0.1))
    }
else:
    weights  = {
        'wc1': tf.Variable(tf.random_normal([5, 5, 3, 128], stdd
ev=0.1)),
        'wc2': tf.Variable(tf.random_normal([5, 5, 128, 128], st
ddev=0.1)),
        'wd1': tf.Variable(tf.random_normal(
                [(int)(imgsize[0]/4*imgsize[1]/4)*128, 128], std
dev=0.1)),
        'wd2': tf.Variable(tf.random_normal([128, n_output], std
dev=0.1))
    }
biases   = {
    'bc1': tf.Variable(tf.random_normal([128], stddev=0.1)),
    'bc2': tf.Variable(tf.random_normal([128], stddev=0.1)),
    'bd1': tf.Variable(tf.random_normal([128], stddev=0.1)),
    'bd2': tf.Variable(tf.random_normal([n_output], stddev=0.1))
}
```

```
def conv_basic(_input, _w, _b, _keepratio, _use_gray):
    # INPUT
    if _use_gray:
        _input_r = tf.reshape(_input, shape=[-1, imgsize[0], img
size[1], 1])
    else:
        _input_r = tf.reshape(_input, shape=[-1, imgsize[0], img
size[1], 3])
    # CONVOLUTION LAYER 1
    _conv1 = tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(_input_r
        , _w['wc1'], strides=[1, 1, 1, 1], padding='SAME'), _b['
bc1']))
    _pool1 = tf.nn.max_pool(_conv1, ksize=[1, 2, 2, 1]
        , strides=[1, 2, 2, 1], padding='SAME')
    _pool_dr1 = tf.nn.dropout(_pool1, _keepratio)
    # CONVOLUTION LAYER 2
    _conv2 = tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(_pool_dr1
        , _w['wc2'], strides=[1, 1, 1, 1], padding='SAME'), _b['
bc2']))
    _pool2 = tf.nn.max_pool(_conv2, ksize=[1, 2, 2, 1]
        , strides=[1, 2, 2, 1], padding='SAME')
    _pool_dr2 = tf.nn.dropout(_pool2, _keepratio)
    # VECTORIZE
    _dense1 = tf.reshape(_pool_dr2
                         , [-1, _w['wd1'].get_shape().as_list()[0
]])
    # FULLY CONNECTED LAYER 1
    _fc1 = tf.nn.relu(tf.add(tf.matmul(_dense1, _w['wd1']), _b['
bd1']))
    _fc_dr1 = tf.nn.dropout(_fc1, _keepratio)
    # FULLY CONNECTED LAYER 2
    _out = tf.add(tf.matmul(_fc_dr1, _w['wd2']), _b['bd2'])
    # RETURN
    out = {
        'input_r': _input_r, 'conv1': _conv1, 'pool1': _pool1
        , 'pool1_dr1': _pool_dr1, 'conv2': _conv2, 'pool2': _poo
l2
        , 'pool_dr2': _pool_dr2, 'dense1': _dense1, 'fc1': _fc1
        , 'fc_dr1': _fc_dr1, 'out': _out
    }
    return out
print ("NETWORK READY")
```

```
NETWORK READY
```

DEFINE FUNCTIONS

```
# tf Graph input
x = tf.placeholder(tf.float32, [None, n_input])
y = tf.placeholder(tf.float32, [None, n_output])
keepratio = tf.placeholder(tf.float32)

# Functions!
_pred = conv_basic(x, weights, biases, keepratio, use_gray)['out'
]
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(_p
red, y))
WEIGHT_DECAY_FACTOR = 0.0001
l2_loss = tf.add_n([tf.nn.l2_loss(v)
            for v in tf.trainable_variables()])
cost = cost + WEIGHT_DECAY_FACTOR*l2_loss
optm = tf.train.AdamOptimizer(learning_rate=0.001).minimize(cost
)
_corr = tf.equal(tf.argmax(_pred,1), tf.argmax(y,1)) # Count cor
rects
accr = tf.reduce_mean(tf.cast(_corr, tf.float32)) # Accuracy
init = tf.initialize_all_variables()
print ("FUNCTIONS READY")
```

```
FUNCTIONS READY
```

OPTIMIZE

```
# Parameters
training_epochs = 400
batch_size      = 100
display_step    = 40

# Launch the graph
sess = tf.Session()
sess.run(init)

# Training cycle
for epoch in range(training_epochs):
    avg_cost = 0.
    num_batch = int(ntrain/batch_size)+1
    # Loop over all batches
    for i in range(num_batch):
        randidx = np.random.randint(ntrain, size=batch_size)
        batch_xs = trainimg[randidx, :]
        batch_ys = trainlabel[randidx, :]
        # Fit training using batch data
        sess.run(optm, feed_dict={x: batch_xs, y: batch_ys
                                 , keepratio:0.7})
        # Compute average loss
        avg_cost += sess.run(cost, feed_dict={x: batch_xs, y: ba
tch_ys
                                , keepratio:1.})/num_batch

    # Display logs per epoch step
    if epoch % display_step == 0 or epoch == training_epochs-1:
        print ("Epoch: %03d/%03d cost: %.9f" %
               (epoch, training_epochs, avg_cost))
        train_acc = sess.run(accr, feed_dict={x: batch_xs
                                , y: batch_ys, keepratio:1.})
        print (" Training accuracy: %.3f" % (train_acc))
        test_acc = sess.run(accr, feed_dict={x: testimg
                                , y: testlabel, keepratio:1.})
        print (" Test accuracy: %.3f" % (test_acc))
print ("Optimization Finished!")
```

```
Epoch: 000/400 cost: 10.054866600
 Training accuracy: 0.750
 Test accuracy: 0.755
Epoch: 040/400 cost: 2.086551237
 Training accuracy: 0.980
 Test accuracy: 0.901
Epoch: 080/400 cost: 1.829108810
 Training accuracy: 1.000
 Test accuracy: 0.930
```

CLOSE SESSION

```
sess.close()
print ("Session closed.")
```

Using Pre-trained Model (VGG)

Use VGG network

```
import scipy.io
import numpy as np
import os
import scipy.misc
import matplotlib.pyplot as plt
import tensorflow as tf
%matplotlib inline
print ("Packages loaded")
```

```
Packages loaded
```

Define network

```
def net(data_path, input_image):
    layers = (
        'conv1_1', 'relu1_1', 'conv1_2', 'relu1_2', 'pool1',
        'conv2_1', 'relu2_1', 'conv2_2', 'relu2_2', 'pool2',
        'conv3_1', 'relu3_1', 'conv3_2', 'relu3_2', 'conv3_3',
        'relu3_3', 'conv3_4', 'relu3_4', 'pool3',
        'conv4_1', 'relu4_1', 'conv4_2', 'relu4_2', 'conv4_3',
        'relu4_3', 'conv4_4', 'relu4_4', 'pool4',
        'conv5_1', 'relu5_1', 'conv5_2', 'relu5_2', 'conv5_3',
        'relu5_3', 'conv5_4', 'relu5_4'
    )
    data = scipy.io.loadmat(data_path)
    mean = data['normalization'][0][0][0]
    mean_pixel = np.mean(mean, axis=(0, 1))
    weights = data['layers'][0]
    net = {}
    current = input_image
    for i, name in enumerate(layers):
        kind = name[:4]
        if kind == 'conv':
            kernels, bias = weights[i][0][0][0][0]
            # matconvnet: weights are [width, height, in_channel
s, out_channels]
            # tensorflow: weights are [height, width, in_channel
s, out_channels]
            kernels = np.transpose(kernels, (1, 0, 2, 3))
            bias = bias.reshape(-1)
            current = _conv_layer(current, kernels, bias)
        elif kind == 'relu':
            current = tf.nn.relu(current)
        elif kind == 'pool':
            current = _pool_layer(current)
        net[name] = current
    assert len(net) == len(layers)
    return net, mean_pixel, layers
print ("Network for VGG ready")
```

```
Network for VGG ready
```

Define functions

```
def _conv_layer(input, weights, bias):
    conv = tf.nn.conv2d(input, tf.constant(weights), strides=(1,
1, 1, 1),
            padding='SAME')
    return tf.nn.bias_add(conv, bias)
def _pool_layer(input):
    return tf.nn.max_pool(input, ksize=(1, 2, 2, 1), strides=(1,
2, 2, 1),
            padding='SAME')
def preprocess(image, mean_pixel):
    return image - mean_pixel
def unprocess(image, mean_pixel):
    return image + mean_pixel
def imread(path):
    return scipy.misc.imread(path).astype(np.float)
def imsave(path, img):
    img = np.clip(img, 0, 255).astype(np.uint8)
    scipy.misc.imsave(path, img)
print ("Functions for VGG ready")
```

```
Functions for VGG ready
```

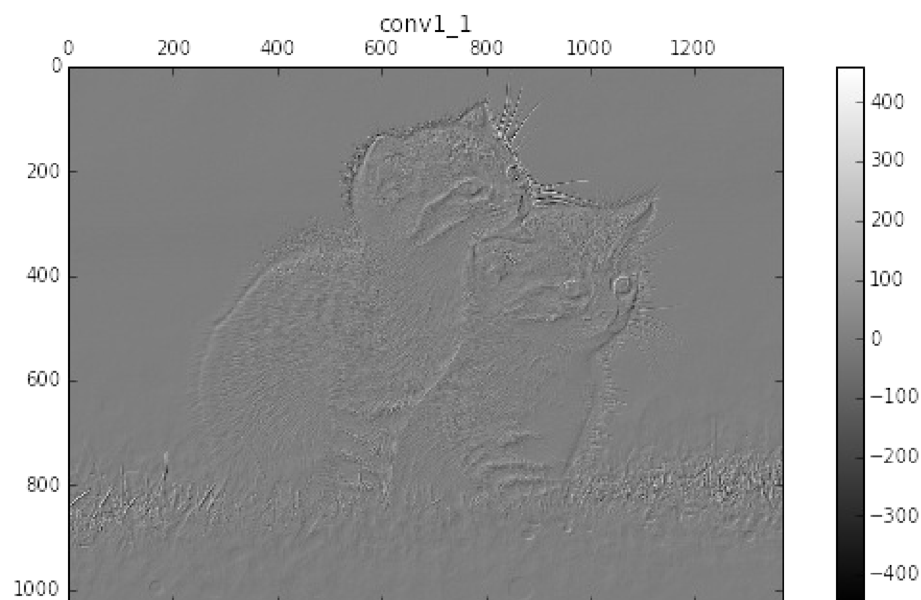
Run

```
cwd  = os.getcwd()
VGG_PATH = cwd + "/data/imagenet-vgg-verydeep-19.mat"
IMG_PATH = cwd + "/images/cat.jpg"
input_image = imread(IMG_PATH)
shape = (1,) + input_image.shape # (h, w, nch) =>  (1, h, w, nch
)
with tf.Graph().as_default(), tf.Session() as sess:
    image = tf.placeholder('float', shape=shape)
    nets, mean_pixel, all_layers = net(VGG_PATH, image)
    input_image_pre = np.array([preprocess(input_image, mean_pix
el)])
    layers = all_layers # For all layers
    # layers = ('relu2_1', 'relu3_1', 'relu4_1')
    for i, layer in enumerate(layers):
        print "[%d/%d] %s" % (i+1, len(layers), layer)
        features = nets[layer].eval(feed_dict={image: input_imag
e_pre})

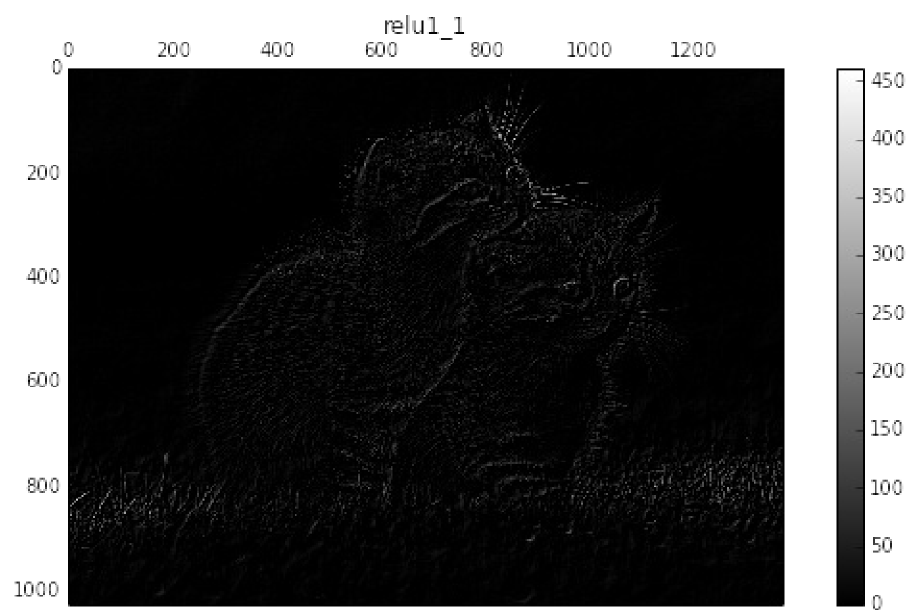
        print " Type of 'features' is ", type(features)
        print " Shape of 'features' is %s" % (features.shape,)
        # Plot response
        if 1:
            plt.figure(i+1, figsize=(10, 5))
            plt.matshow(features[0, :, :, 0], cmap=plt.cm.gray,
fignum=i+1)
            plt.title("" + layer)
            plt.colorbar()
            plt.show()
```

```
[1/36] conv1_1
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 1026, 1368, 64)

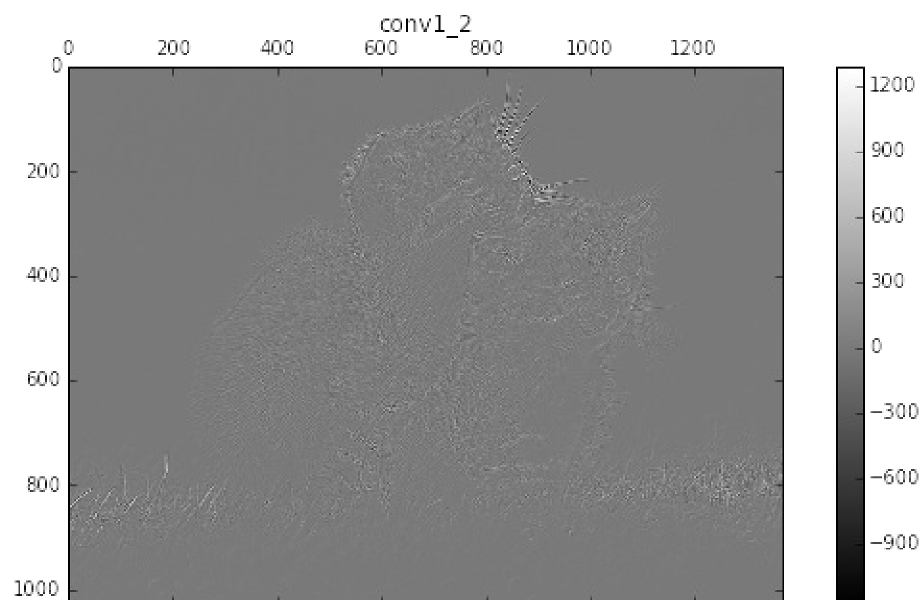

/usr/lib/pymodules/python2.7/matplotlib/collections.py:548: Futu
reWarning: elementwise comparison failed; returning scalar inste
ad, but in the future will perform elementwise comparison
  if self._edgecolors == 'face':
```



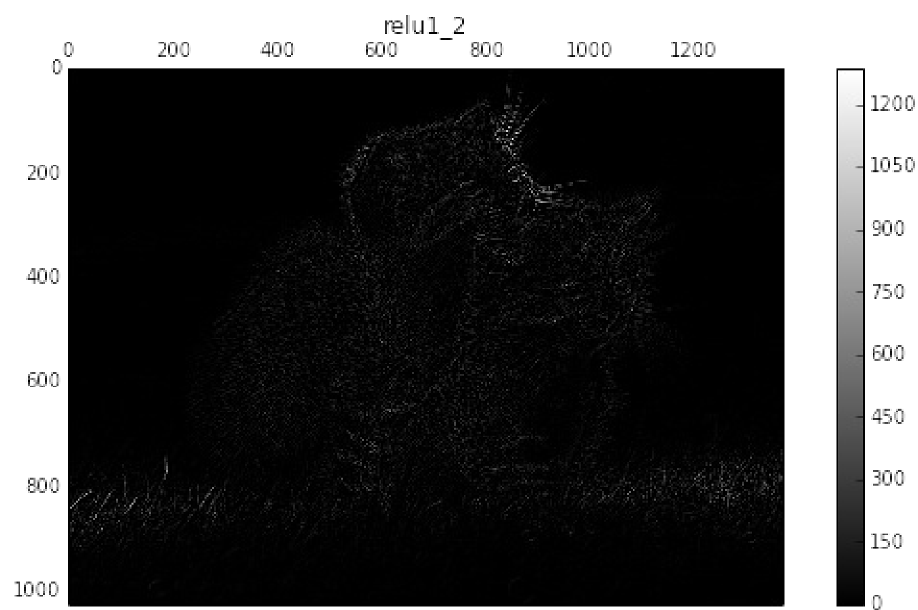

```
[2/36] relu1_1
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 1026, 1368, 64)
```



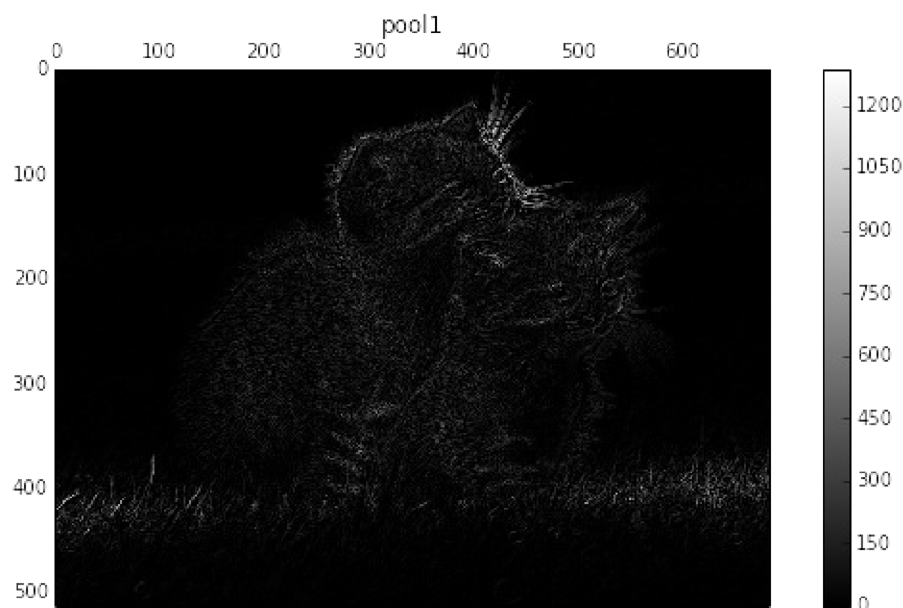

```
[3/36] conv1_2
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 1026, 1368, 64)
```



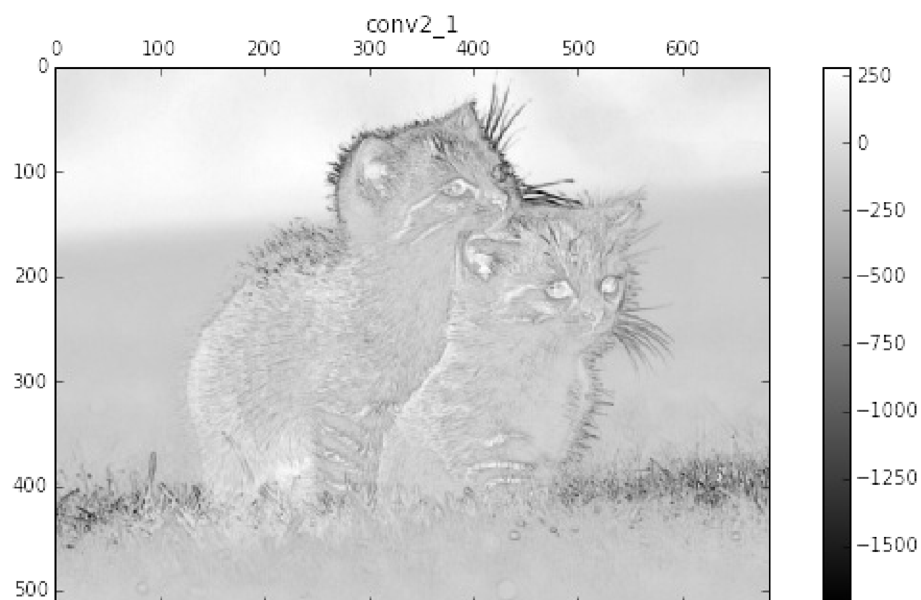

```
[4/36] relu1_2
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 1026, 1368, 64)
```



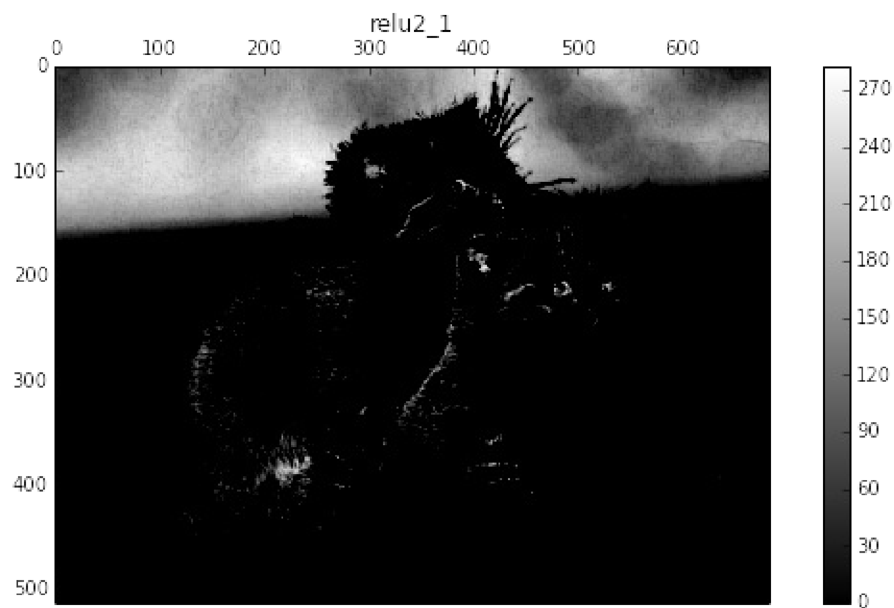

```
[5/36] pool1
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 513, 684, 64)
```



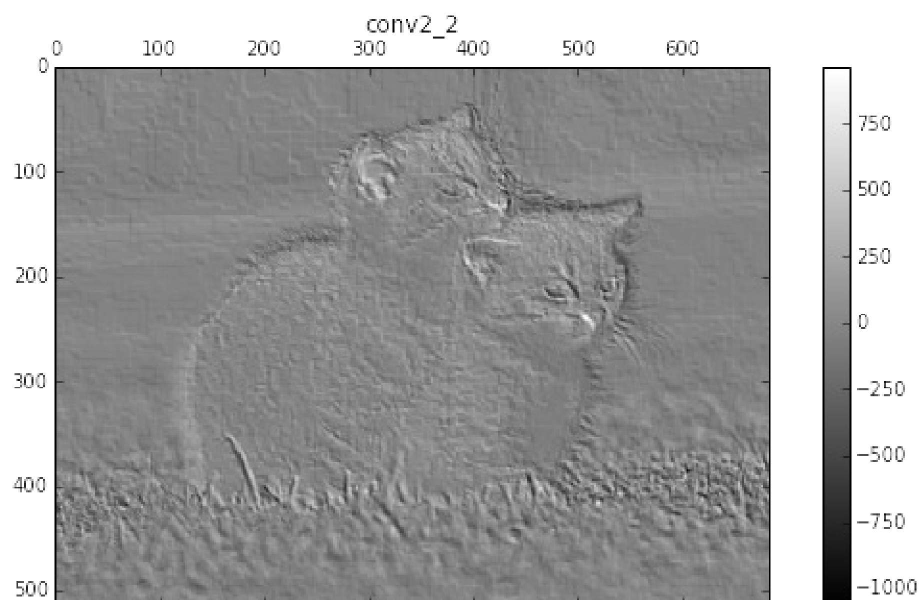

```
[6/36] conv2_1
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 513, 684, 128)
```



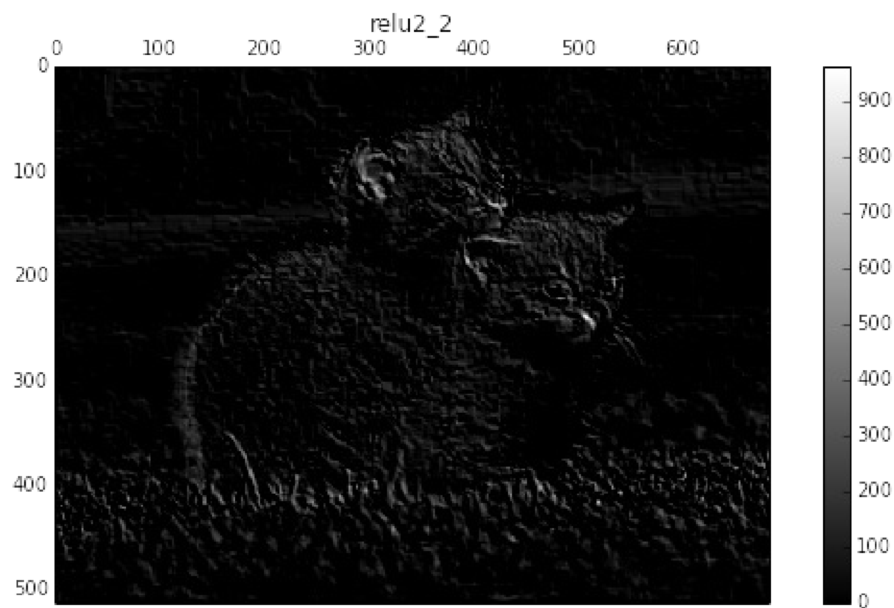

```
[7/36] relu2_1
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 513, 684, 128)
```



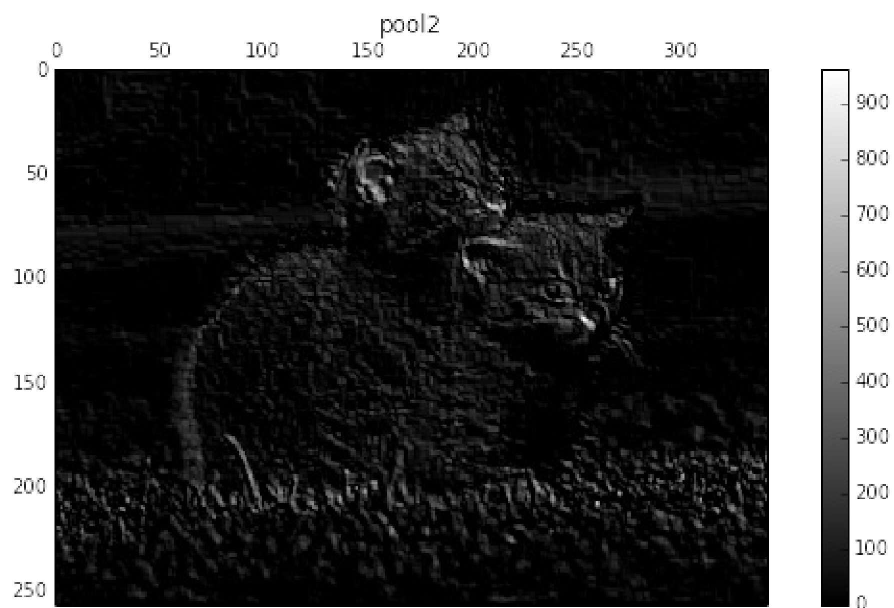

```
[8/36] conv2_2
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 513, 684, 128)
```



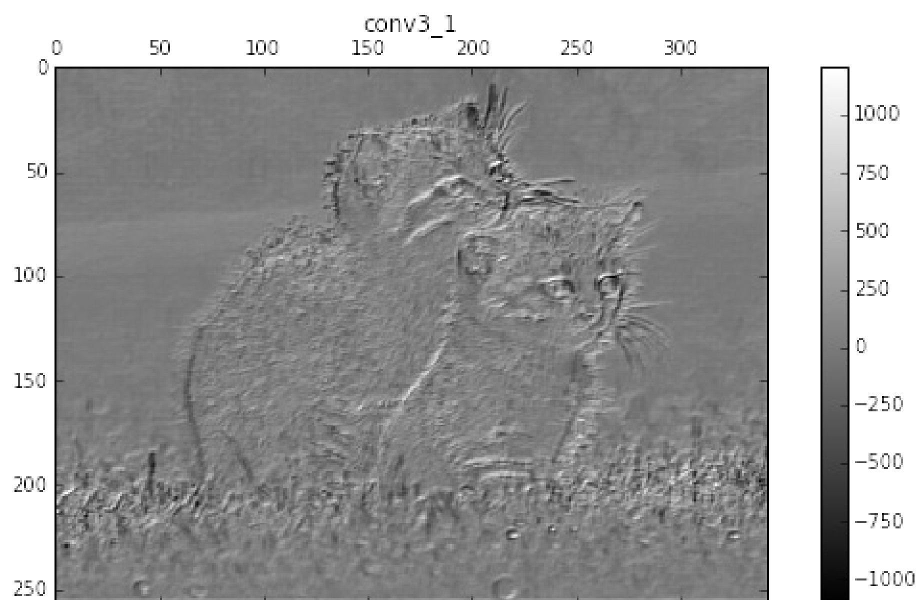

```
[9/36] relu2_2
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 513, 684, 128)
```



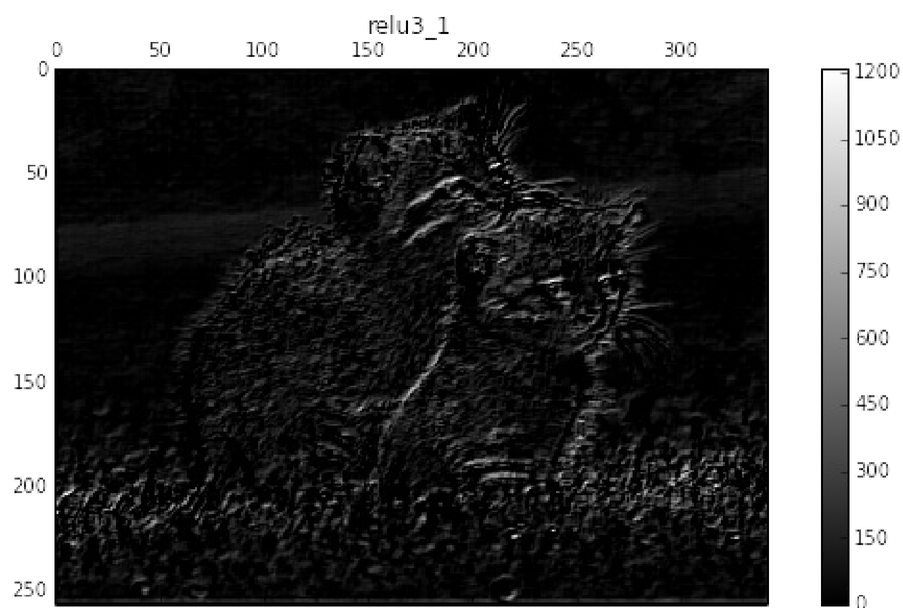

```
[10/36] pool2
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 257, 342, 128)
```



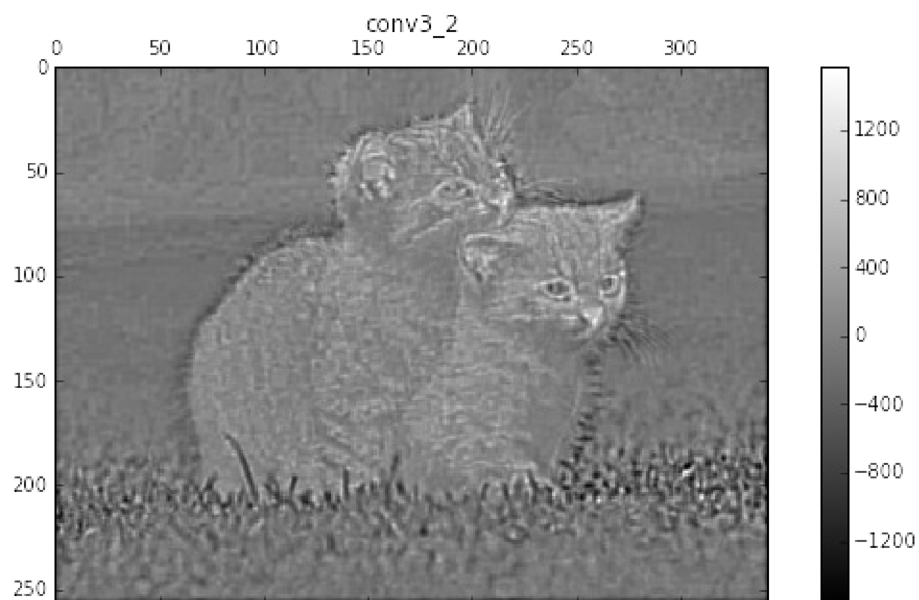

```
[11/36] conv3_1
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 257, 342, 256)
```



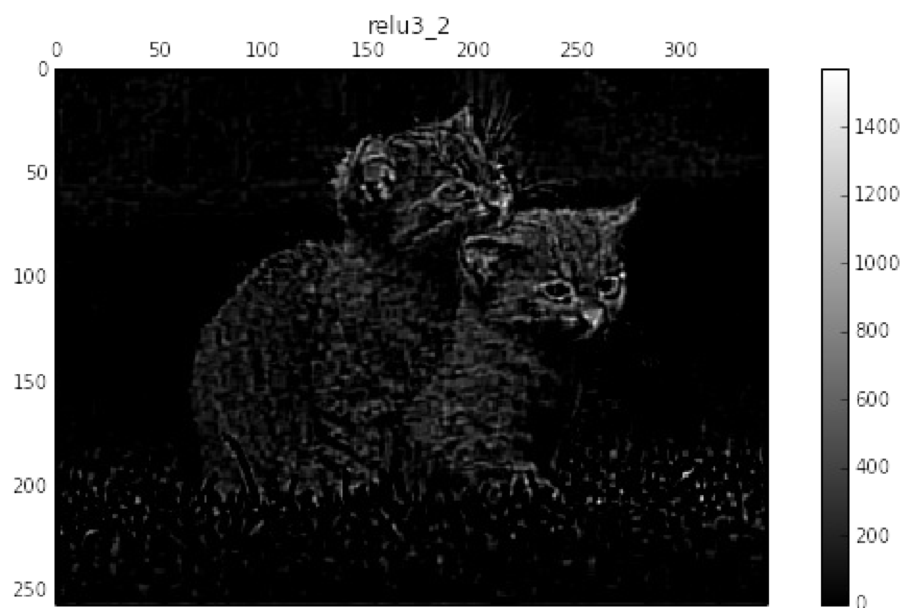

```
[12/36] relu3_1
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 257, 342, 256)
```



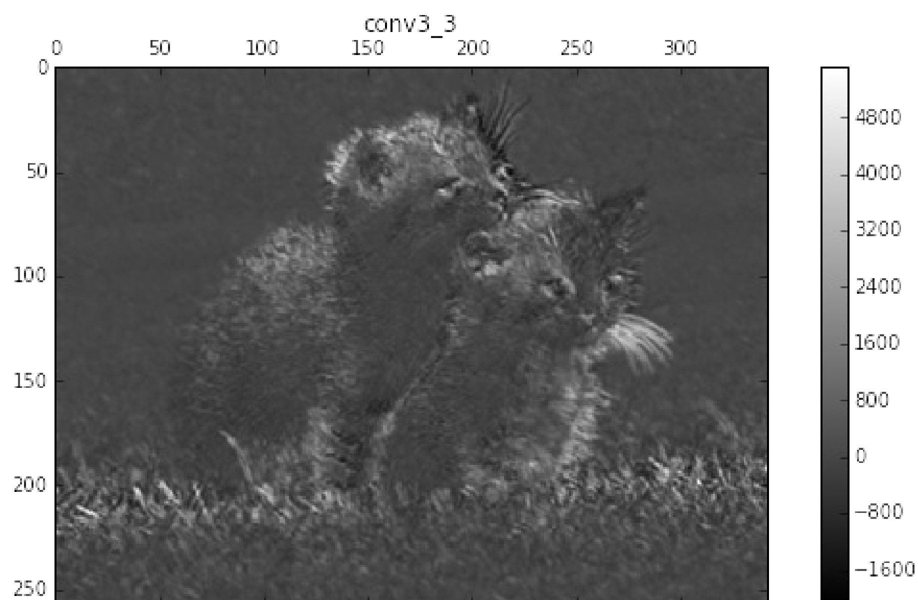

```
[13/36] conv3_2
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 257, 342, 256)
```



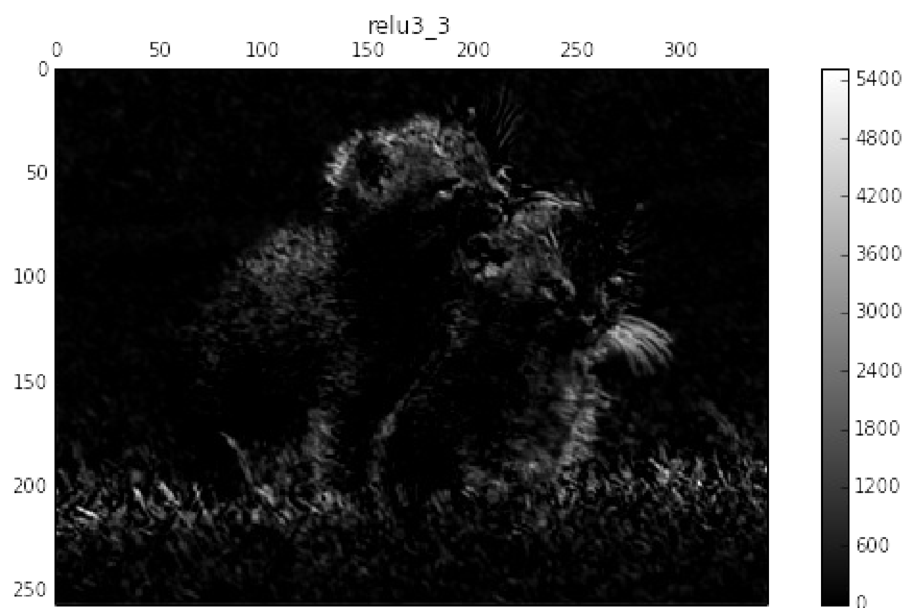

```
[14/36] relu3_2
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 257, 342, 256)
```



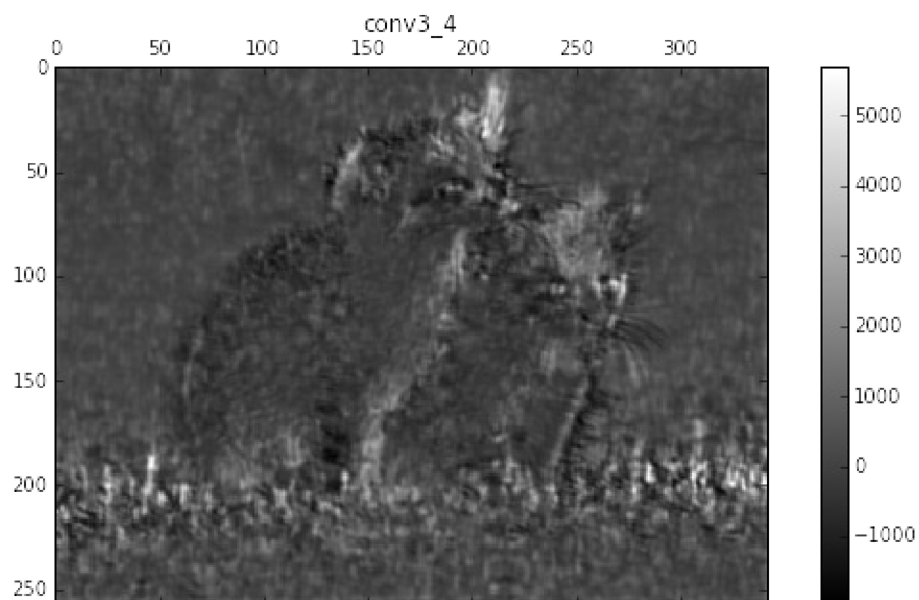

```
[15/36] conv3_3
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 257, 342, 256)
```



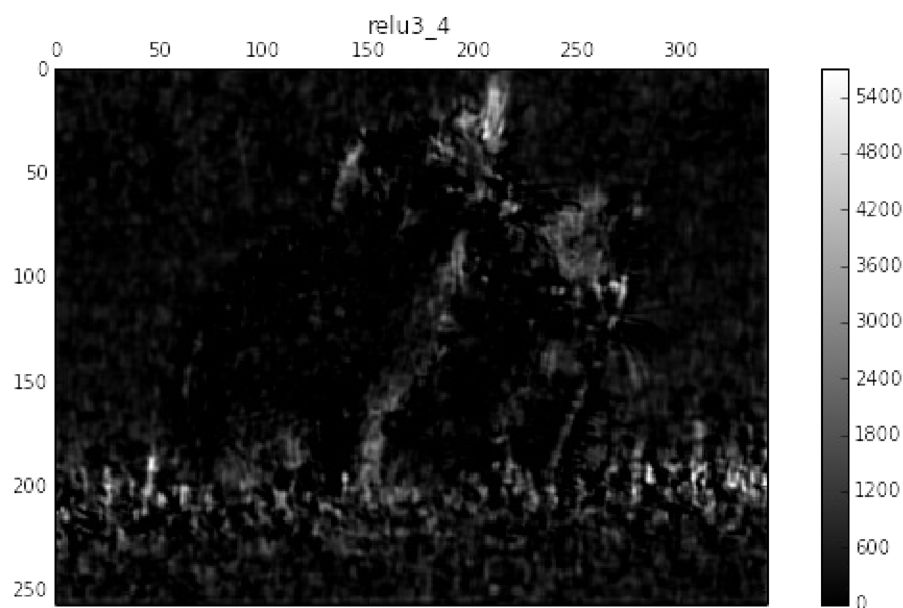

```
[16/36] relu3_3
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 257, 342, 256)
```



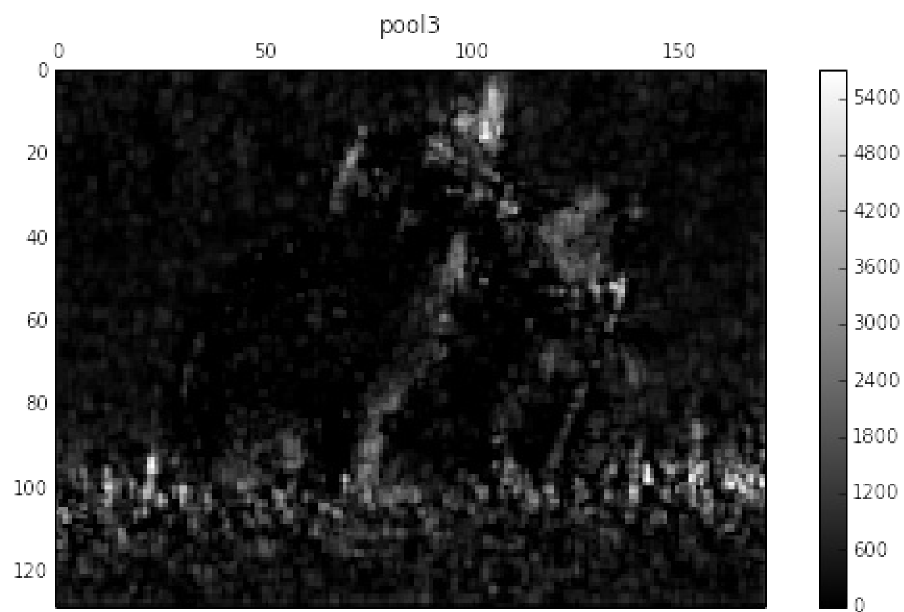

```
[17/36] conv3_4
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 257, 342, 256)
```



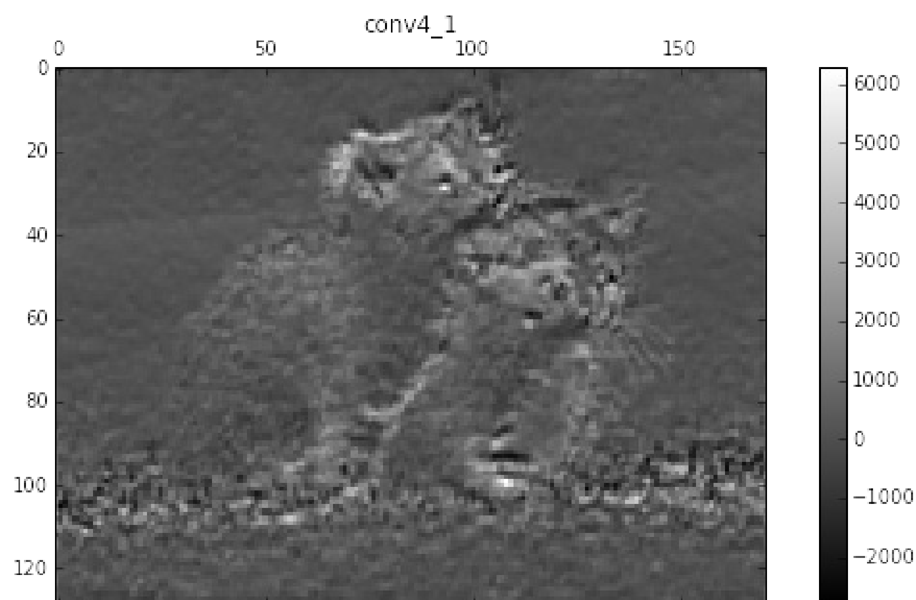

```
[18/36] relu3_4
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 257, 342, 256)
```



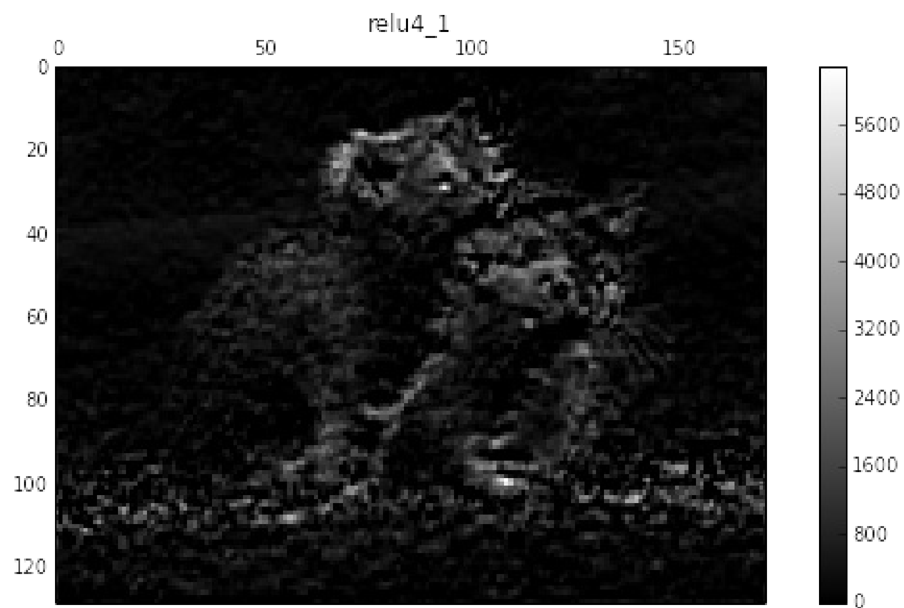

```
[19/36] pool3
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 129, 171, 256)
```



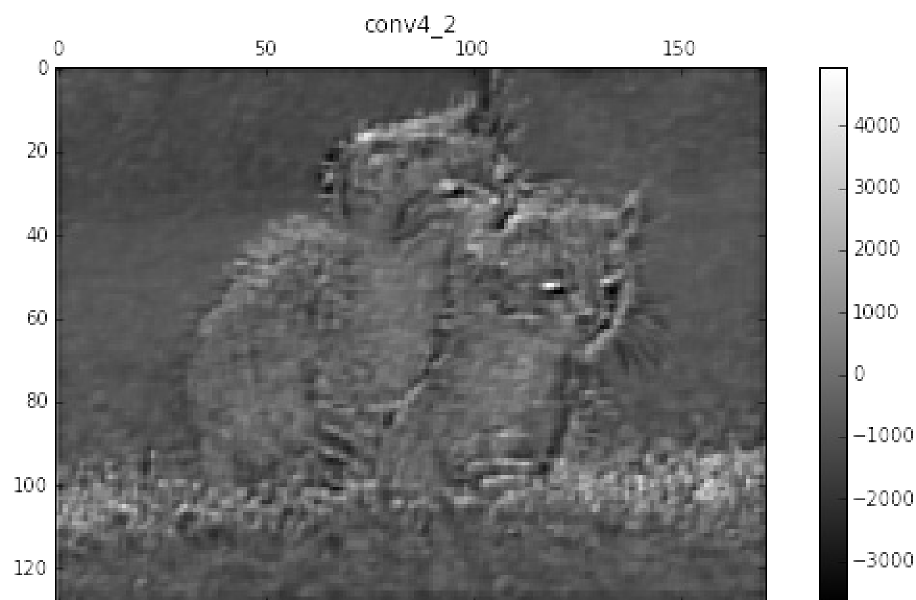

```
[20/36] conv4_1
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 129, 171, 512)
```



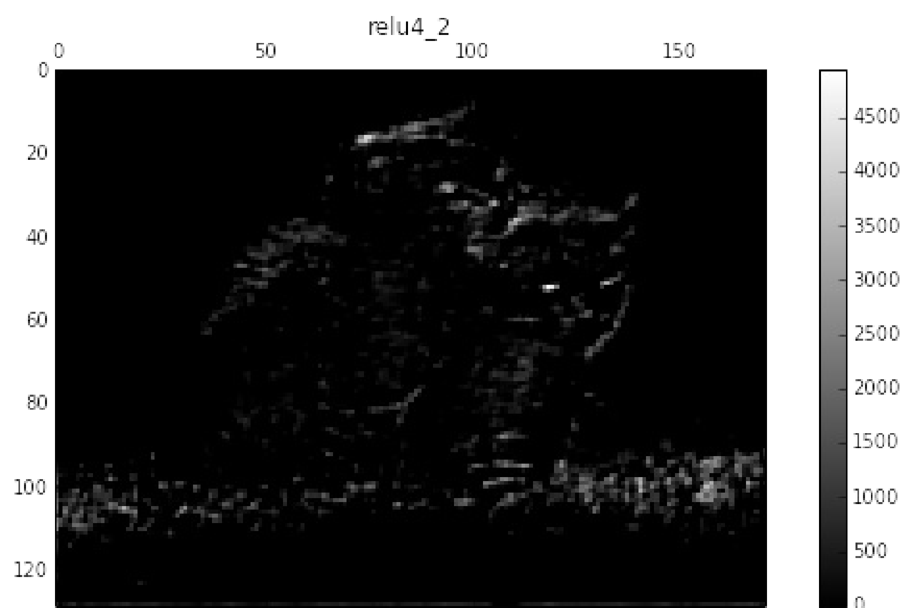

```
[21/36] relu4_1
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 129, 171, 512)
```



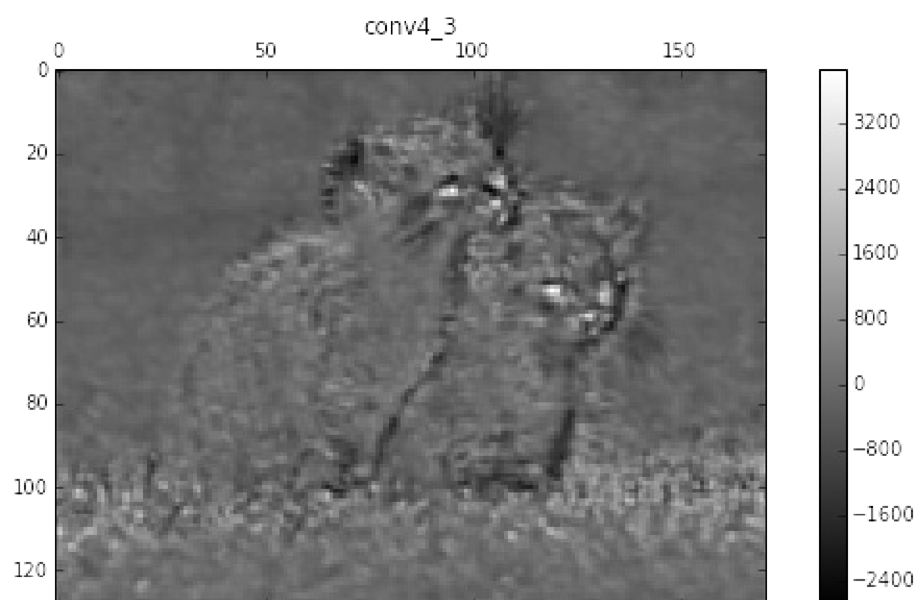

```
[22/36] conv4_2
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 129, 171, 512)
```



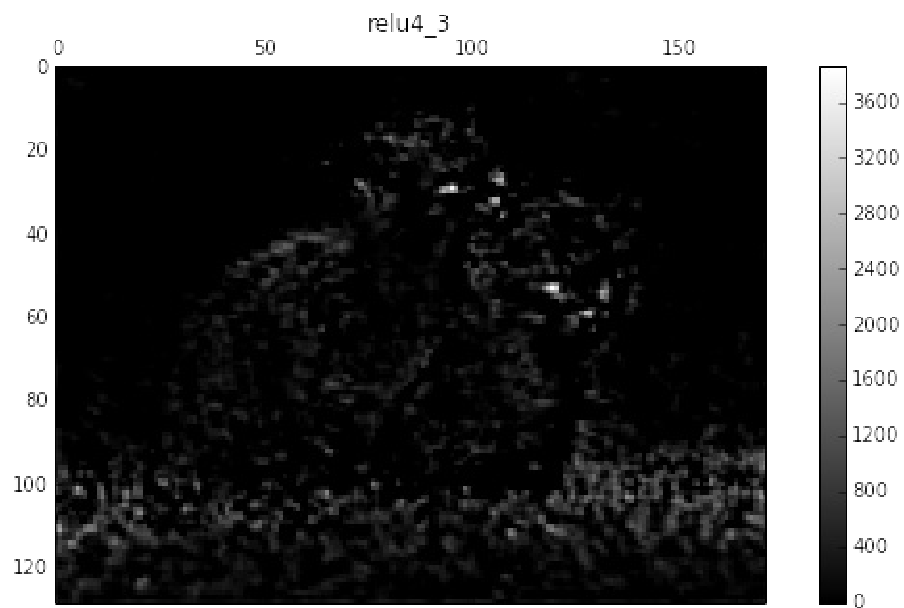

```
[23/36] relu4_2
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 129, 171, 512)
```



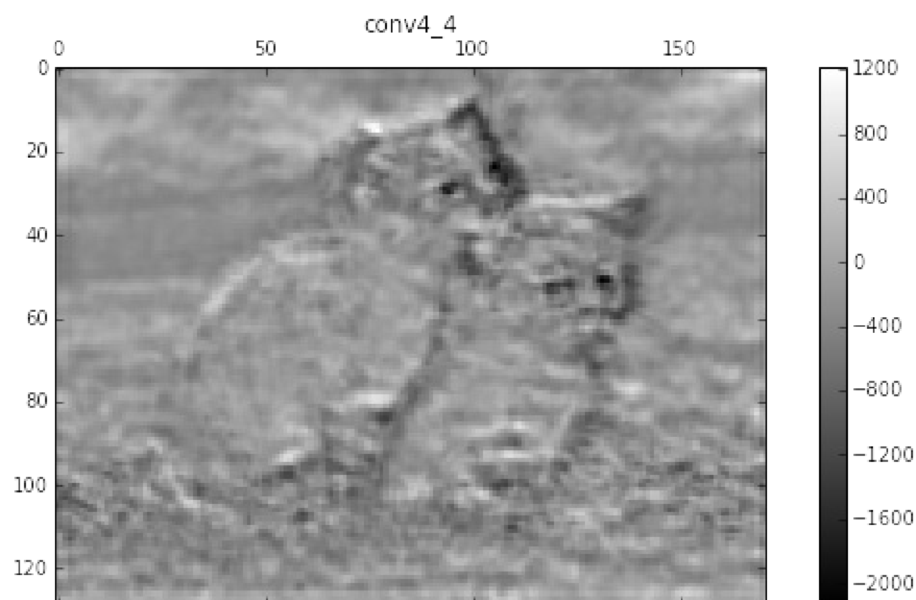

```
[24/36] conv4_3
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 129, 171, 512)
```



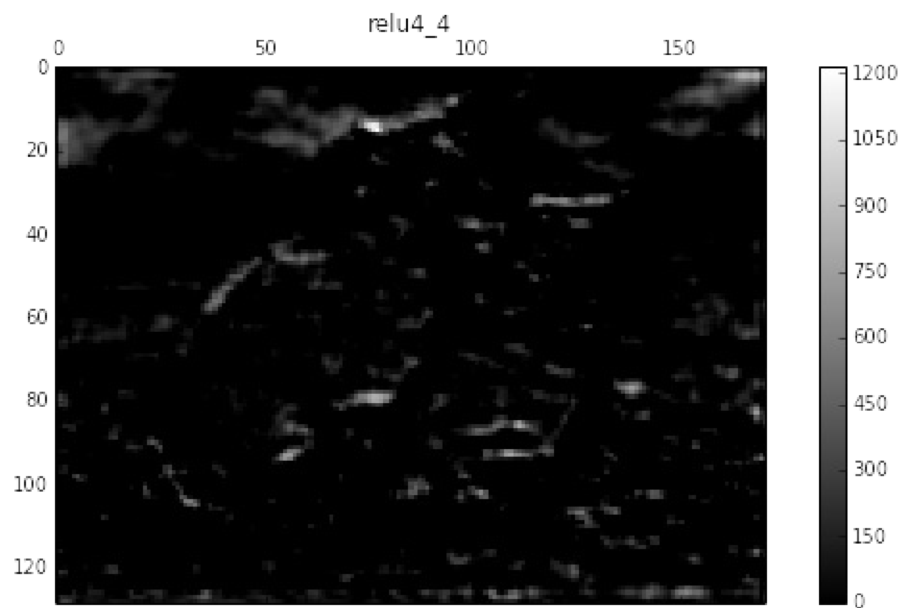

```
[25/36] relu4_3
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 129, 171, 512)
```



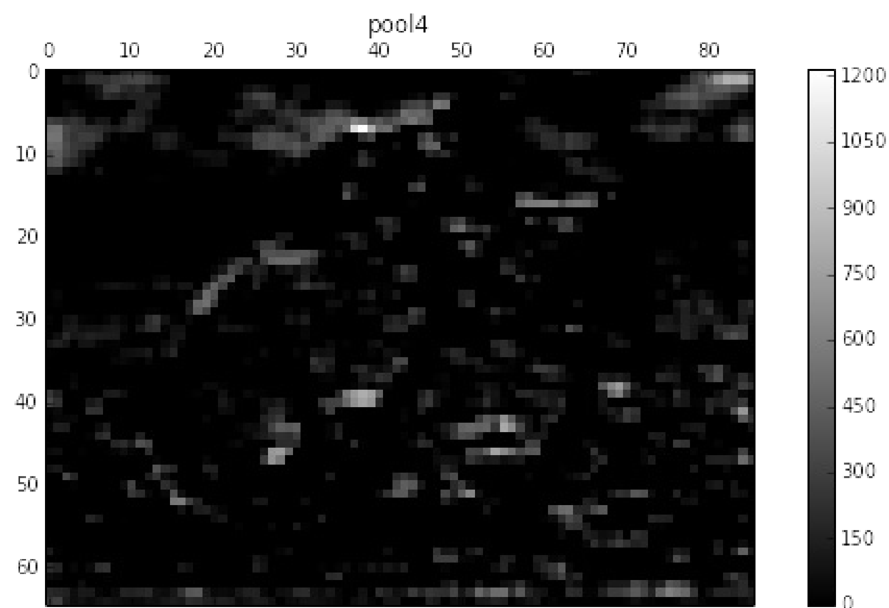

```
[26/36] conv4_4
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 129, 171, 512)
```



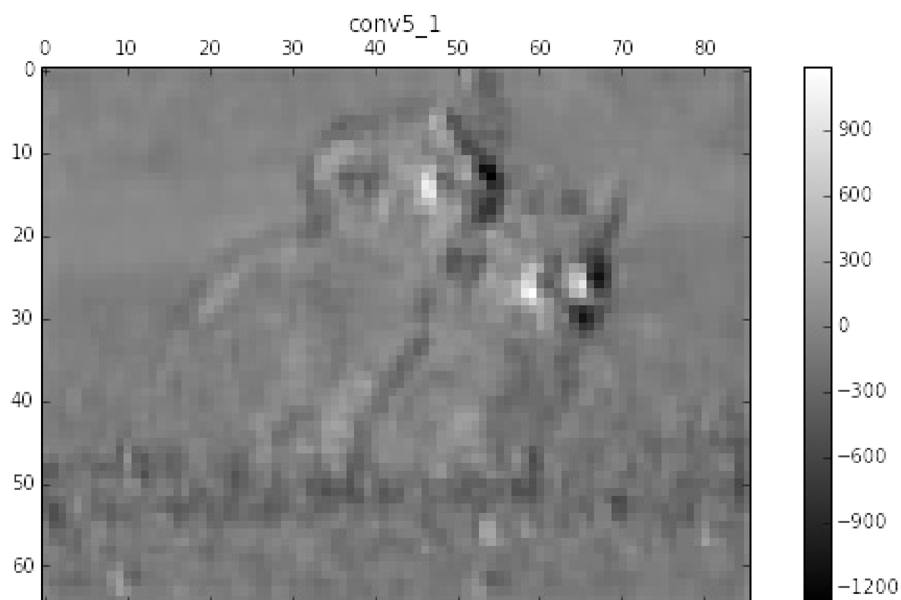

```
[27/36] relu4_4
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 129, 171, 512)
```



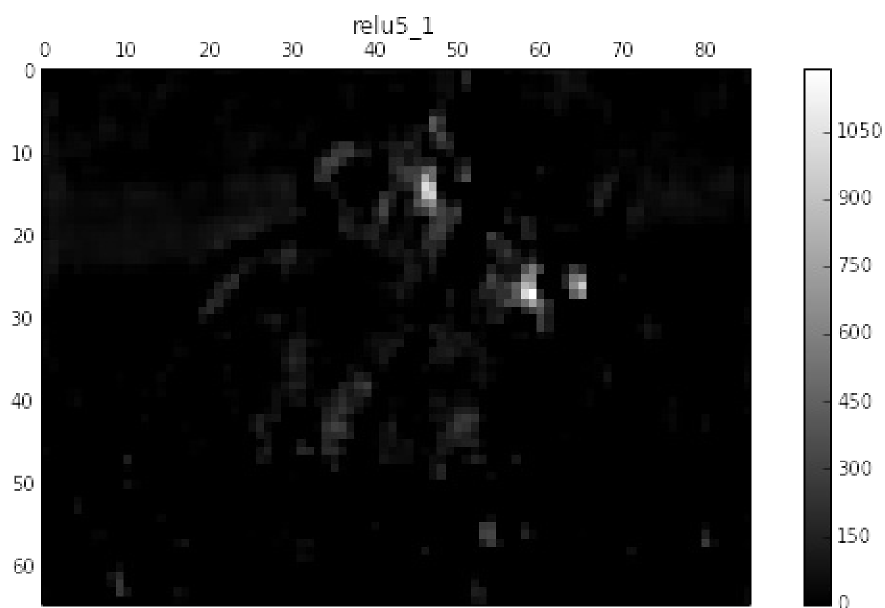

```
[28/36] pool4
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 65, 86, 512)
```



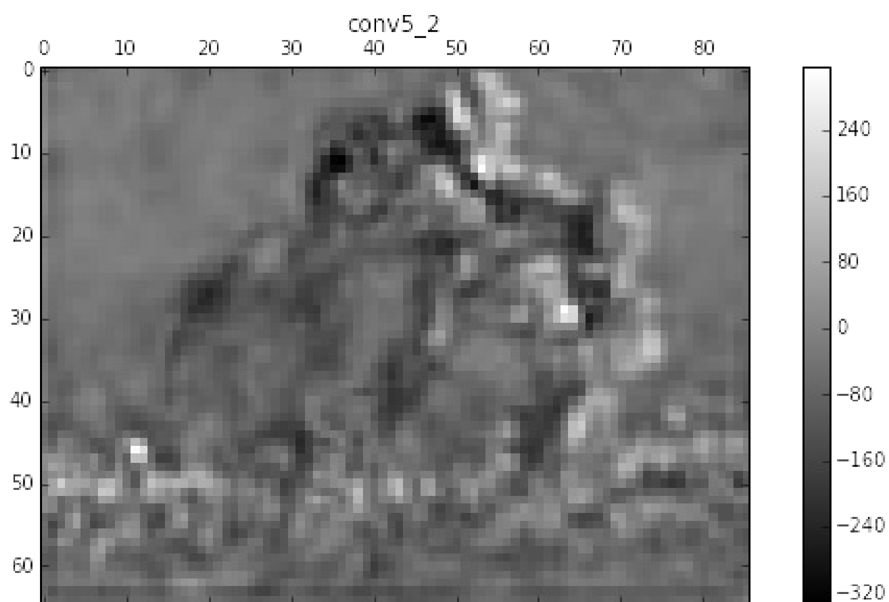

```
[29/36] conv5_1
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 65, 86, 512)
```



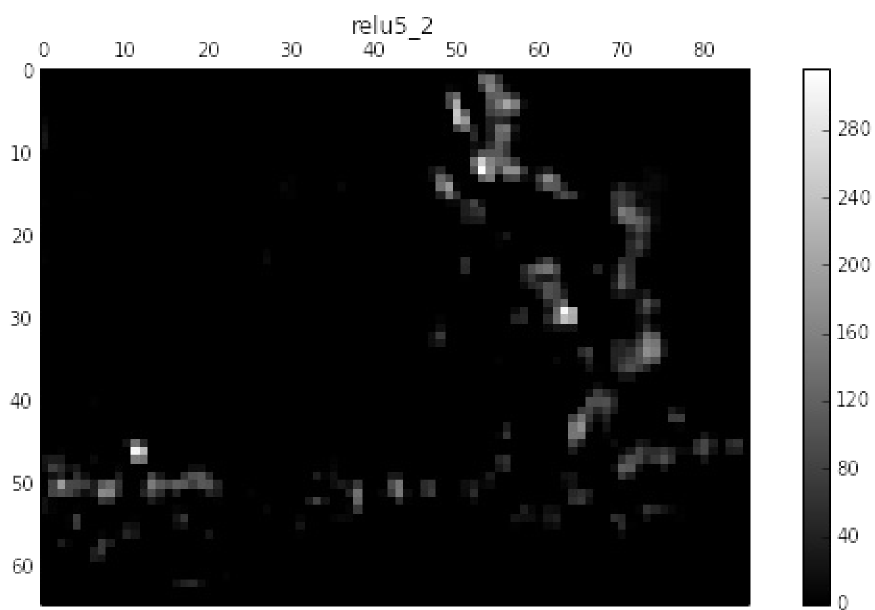

```
[30/36] relu5_1
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 65, 86, 512)
```



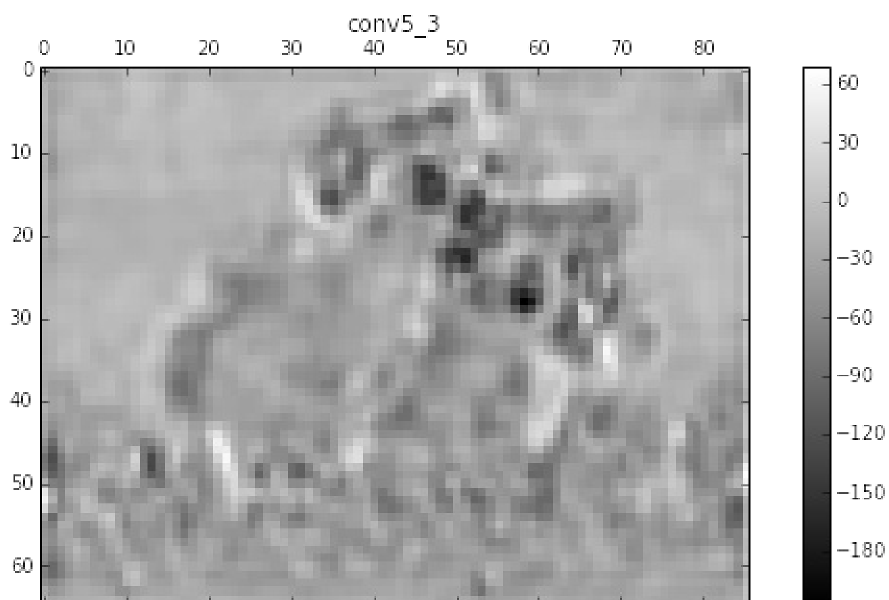

```
[31/36] conv5_2
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 65, 86, 512)
```



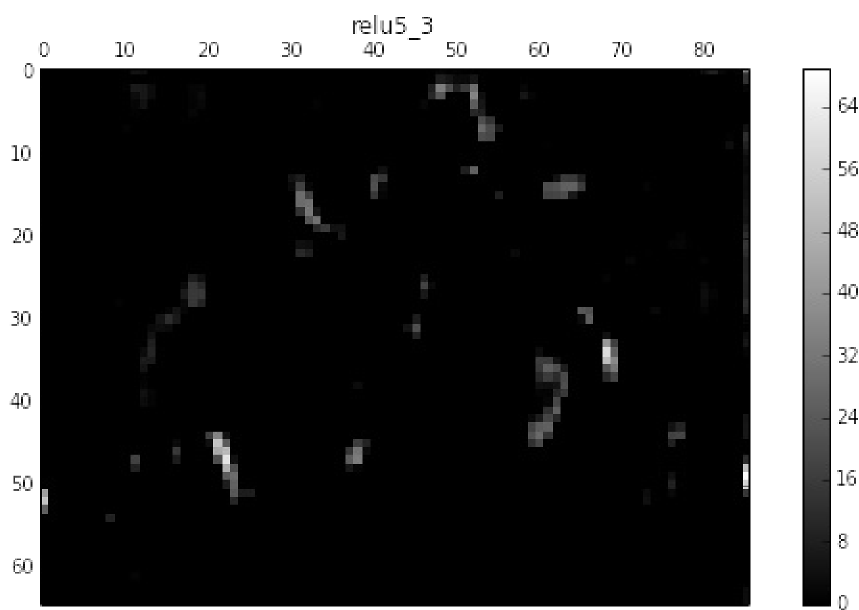

```
[32/36] relu5_2
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 65, 86, 512)
```



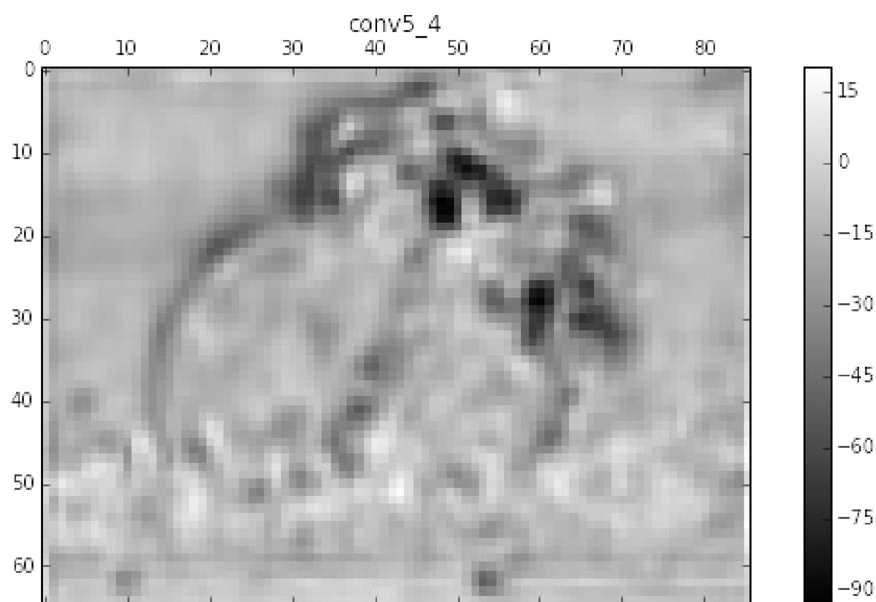

```
[33/36] conv5_3
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 65, 86, 512)
```



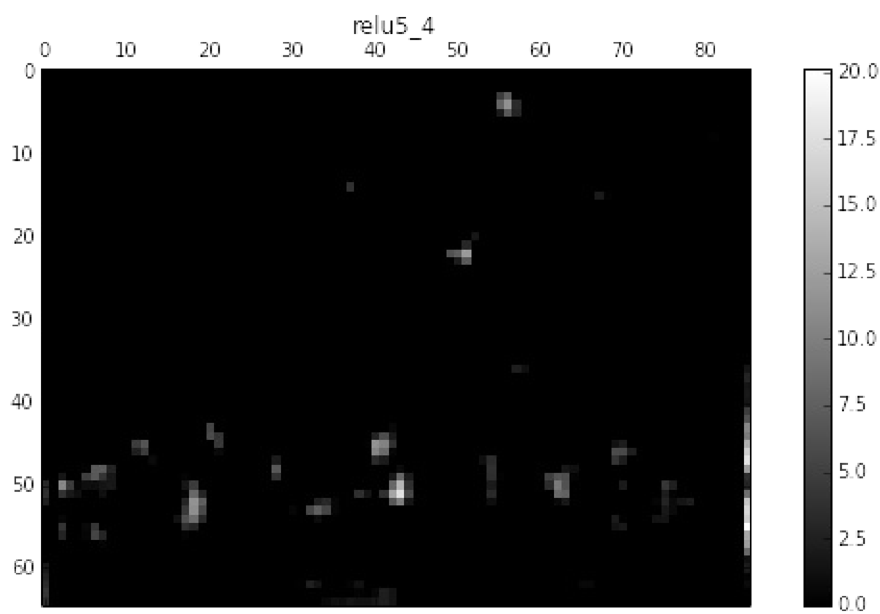

```
[34/36] relu5_3
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 65, 86, 512)
```




```
[35/36] conv5_4
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 65, 86, 512)
```




```
[36/36] relu5_4
 Type of 'features' is  <type 'numpy.ndarray'>
 Shape of 'features' is (1, 65, 86, 512)
```



CNN FINETUNING WITH PRE-TRAINED VGG NET

```
import os
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import scipy.misc
import scipy.io
from tensorflow.examples.tutorials.mnist import input_data
%matplotlib inline
print ("Packages loaded.")
```

```
Packages loaded.
```

LOAD DATA

```
cwd = os.getcwd()
loadpath = cwd + "/data/data4vgg.npz"
l = np.load(loadpath)

# Show Files
print (l.files)
```

```
['trainlabel', 'trainimg', 'testimg', 'testlabel']
```

PARSE DATA

```
trainimg   = l['trainimg']
trainlabel = l['trainlabel']
testimg    = l['testimg']
testlabel  = l['testlabel']
ntrain     = trainimg.shape[0]
nclass     = trainlabel.shape[1]
dim        = trainimg.shape[1]
ntest      = testimg.shape[0]

print ("%d train images loaded" % (ntrain))
print ("%d test images loaded"  % (ntest))
print ("%d dimensional input"   % (dim))
print ("%d classes"             % (nclass))
print ("shape of 'trainimg' is %s" % (trainimg.shape,))
print ("shape of 'testimg' is %s" % (testimg.shape,))
```

```
69 train images loaded
18 test images loaded
37632 dimensional input
2 classes
shape of 'trainimg' is (69, 37632)
shape of 'testimg' is (18, 37632)
```

GENERATE TENSORS FOR TRAINING AND TESTING

```
trainimg_tensor = np.ndarray((ntrain, 112, 112, 3))
for i in range(ntrain):
    currimg = trainimg[i, :]
    currimg = np.reshape(currimg, [112, 112, 3])
    trainimg_tensor[i, :, :, :] = currimg
print ("shape of trainimg_tensor is %s" % (trainimg_tensor.shape
,))

testimg_tensor = np.ndarray((ntest, 112, 112, 3))
for i in range(ntest):
    currimg = testimg[i, :]
    currimg = np.reshape(currimg, [112, 112, 3])
    testimg_tensor[i, :, :, :] = currimg
print ("shape of testimg_tensor is %s" % (testimg_tensor.shape,)
)
```

```
shape of trainimg_tensor is (69, 112, 112, 3)
shape of testimg_tensor is (18, 112, 112, 3)
```

DEFINE A FUNCTION FOR USING PRETRAINED VGG NETWORK

```
def net(data_path, input_image):
    layers = (
        'conv1_1', 'relu1_1', 'conv1_2', 'relu1_2', 'pool1',
        'conv2_1', 'relu2_1', 'conv2_2', 'relu2_2', 'pool2',
        'conv3_1', 'relu3_1', 'conv3_2', 'relu3_2', 'conv3_3',
        'relu3_3', 'conv3_4', 'relu3_4', 'pool3',
        'conv4_1', 'relu4_1', 'conv4_2', 'relu4_2', 'conv4_3',
        'relu4_3', 'conv4_4', 'relu4_4', 'pool4',
        'conv5_1', 'relu5_1', 'conv5_2', 'relu5_2', 'conv5_3',
        'relu5_3', 'conv5_4', 'relu5_4'
    )
    data = scipy.io.loadmat(data_path)
    mean = data['normalization'][0][0][0]
    mean_pixel = np.mean(mean, axis=(0, 1))
    weights = data['layers'][0]
    net = {}
    current = input_image
    for i, name in enumerate(layers):
        kind = name[:4]
        if kind == 'conv':
            kernels, bias = weights[i][0][0][0][0]
            # matconvnet: weights are [width, height, in_channel
s, out_channels]
            # tensorflow: weights are [height, width, in_channel
s, out_channels]
            kernels = np.transpose(kernels, (1, 0, 2, 3))
            bias = bias.reshape(-1)
            current = _conv_layer(current, kernels, bias)
        elif kind == 'relu':
            current = tf.nn.relu(current)
        elif kind == 'pool':
            current = _pool_layer(current)
        net[name] = current

    assert len(net) == len(layers)
    return net, mean_pixel
def _conv_layer(input, weights, bias):
    conv = tf.nn.conv2d(input, tf.constant(weights), strides=(1,
1, 1, 1),
            padding='SAME')
    return tf.nn.bias_add(conv, bias)
def _pool_layer(input):
    return tf.nn.max_pool(input, ksize=(1, 2, 2, 1), strides=(1,
2, 2, 1),
            padding='SAME')
def preprocess(image, mean_pixel):
    return image - mean_pixel
def unprocess(image, mean_pixel):
    return image + mean_pixel
```

EXTRACT FEATURES FROM THE VGG NETWORK

```
VGG_PATH = cwd + "/data/imagenet-vgg-verydeep-19.mat"
with tf.Graph().as_default(), tf.Session() as sess:
    with tf.device("/cpu:0"):
        img_placeholder = tf.placeholder(tf.float32, shape=(None
, 112, 112, 3))
        net_val, mean_pixel = net(VGG_PATH, img_placeholder)
        train_features = net_val['relu5_4'].eval(feed_dict={img_
placeholder: trainimg_tensor})
        test_features = net_val['relu5_4'].eval(feed_dict={img_p
laceholder: testimg_tensor})
print ("TYPE OF 'train_features' IS %s" % (type(train_features))
)
print ("SHAPE OF 'train_features' IS %s" % (train_features.shape
,))
print ("TYPE OF 'test_features' IS %s" % (type(test_features)))
print ("SHAPE OF 'test_features' IS %s" % (test_features.shape,)
)
print("PREPROCESSING DONE")
```

```
TYPE OF 'train_features' IS <type 'numpy.ndarray'>
SHAPE OF 'train_features' IS (69, 7, 7, 512)
TYPE OF 'test_features' IS <type 'numpy.ndarray'>
SHAPE OF 'test_features' IS (18, 7, 7, 512)
PREPROCESSING DONE
```

VECTORIZE CNN FEATURES

```
train_vectorized = np.ndarray((ntrain, 7*7*512))
test_vectorized  = np.ndarray((ntest, 7*7*512))
for i in range(ntrain):
    curr_feat = train_features[i, :, :, :]
    curr_feat_vec = np.reshape(curr_feat, (1, -1))
    train_vectorized[i, :] = curr_feat_vec

for i in range(ntest):
    curr_feat = test_features[i, :, :, :]
    curr_feat_vec = np.reshape(curr_feat, (1, -1))
    test_vectorized[i, :] = curr_feat_vec

print ("SHAPE OF 'train_vectorized' IS %s" % (train_vectorized.s
hape,))
print ("SHAPE OF 'test_vectorized' IS %s" % (test_vectorized.sha
pe,))
```

```
SHAPE OF 'train_vectorized' IS (69, 25088)
SHAPE OF 'test_vectorized' IS (18, 25088)
```

DEFINE NETWORKS AND FUNCTIONS (ADD 2LAYER MLP)

```
# Parameters
learning_rate   = 0.0001
training_epochs = 100
batch_size      = 100
display_step    = 10

# Network
with tf.device("/cpu:0"):
    n_input  = dim
    n_output = nclass
    weights  = {
        'wd1': tf.Variable(tf.random_normal([7*7*512, 1024], std
dev=0.1)),
        'wd2': tf.Variable(tf.random_normal([1024, n_output], st
ddev=0.1))
    }
    biases   = {
        'bd1': tf.Variable(tf.random_normal([1024], stddev=0.1))
,
        'bd2': tf.Variable(tf.random_normal([n_output], stddev=0
.1))
    }
    def conv_basic(_input, _w, _b, _keepratio):
```

```
        # Input
        _input_r = _input
        # Vectorize
        _dense1 = tf.reshape(_input_r, [-1, _w['wd1'].get_shape(
).as_list()[0]])
        # Fc1
        _fc1 = tf.nn.relu(tf.add(tf.matmul(_dense1, _w['wd1']),
_b['bd1']))
        _fc_dr1 = tf.nn.dropout(_fc1, _keepratio)
        # Fc2
        _out = tf.add(tf.matmul(_fc_dr1, _w['wd2']), _b['bd2'])
        # Return everything
        out = {'input_r': _input_r, 'dense1': _dense1,
            'fc1': _fc1, 'fc_dr1': _fc_dr1, 'out': _out }
        return out

# tf Graph input
x = tf.placeholder(tf.float32, [None, 7*7*512])
y = tf.placeholder(tf.float32, [None, n_output])
keepratio = tf.placeholder(tf.float32)

# Functions!
with tf.device("/cpu:0"):
    _pred = conv_basic(x, weights, biases, keepratio)['out']
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logit
s(_pred, y))
    optm = tf.train.AdamOptimizer(learning_rate=learning_rate).m
inimize(cost)
    _corr = tf.equal(tf.argmax(_pred,1), tf.argmax(y,1)) # Count
 corrects
    accr = tf.reduce_mean(tf.cast(_corr, tf.float32)) # Accuracy
    init = tf.initialize_all_variables()

print ("Network Ready to Go!")
```

```
Network Ready to Go!
```

CNN FINETUNING

```
sess = tf.Session()
sess.run(init)

# Training cycle
for epoch in range(training_epochs):
    avg_cost = 0.
    num_batch = int(ntrain/batch_size)+1
    # Loop over all batches
    for i in range(num_batch):
        randidx  = np.random.randint(ntrain, size=batch_size)
        batch_xs = train_vectorized[randidx, :]
        batch_ys = trainlabel[randidx, :]
        # Fit training using batch data
        sess.run(optm, feed_dict={x: batch_xs, y: batch_ys, keep
ratio:0.7})
        # Compute average loss
        avg_cost += sess.run(cost, feed_dict={x: batch_xs, y: ba
tch_ys, keepratio:1.})/num_batch

    # Display logs per epoch step
    if epoch % display_step == 0:
        print ("Epoch: %03d/%03d cost: %.9f" % (epoch, training_
epochs, avg_cost))
        train_acc = sess.run(accr, feed_dict={x: batch_xs, y: ba
tch_ys, keepratio:1.})
        print (" Training accuracy: %.3f" % (train_acc))
        test_acc = sess.run(accr, feed_dict={x: test_vectorized,
 y: testlabel, keepratio:1.})
        print (" Test accuracy: %.3f" % (test_acc))

print ("Optimization Finished!")
```

```
Epoch: 000/100 cost: 3.248429298
 Training accuracy: 0.760
 Test accuracy: 0.444
Epoch: 010/100 cost: 0.146418720
 Training accuracy: 0.980
 Test accuracy: 0.722
Epoch: 020/100 cost: 0.000040202
 Training accuracy: 1.000
 Test accuracy: 0.778
Epoch: 030/100 cost: 0.000000000
 Training accuracy: 1.000
 Test accuracy: 0.778
Epoch: 040/100 cost: 0.000000000
 Training accuracy: 1.000
 Test accuracy: 0.778
Epoch: 050/100 cost: 0.000000000
 Training accuracy: 1.000
 Test accuracy: 0.778
Epoch: 060/100 cost: 0.000000000
 Training accuracy: 1.000
 Test accuracy: 0.722
Epoch: 070/100 cost: 0.000000000
 Training accuracy: 1.000
 Test accuracy: 0.722
Epoch: 080/100 cost: 0.000000000
 Training accuracy: 1.000
 Test accuracy: 0.778
Epoch: 090/100 cost: 0.000000000
 Training accuracy: 1.000
 Test accuracy: 0.889
Optimization Finished!
```

Recurrent Neural Network (RNN)

Sequence classification with LSTM

```
import tensorflow as tf
import tensorflow.examples.tutorials.mnist.input_data as input_d
ata
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
print ("Packages imported")

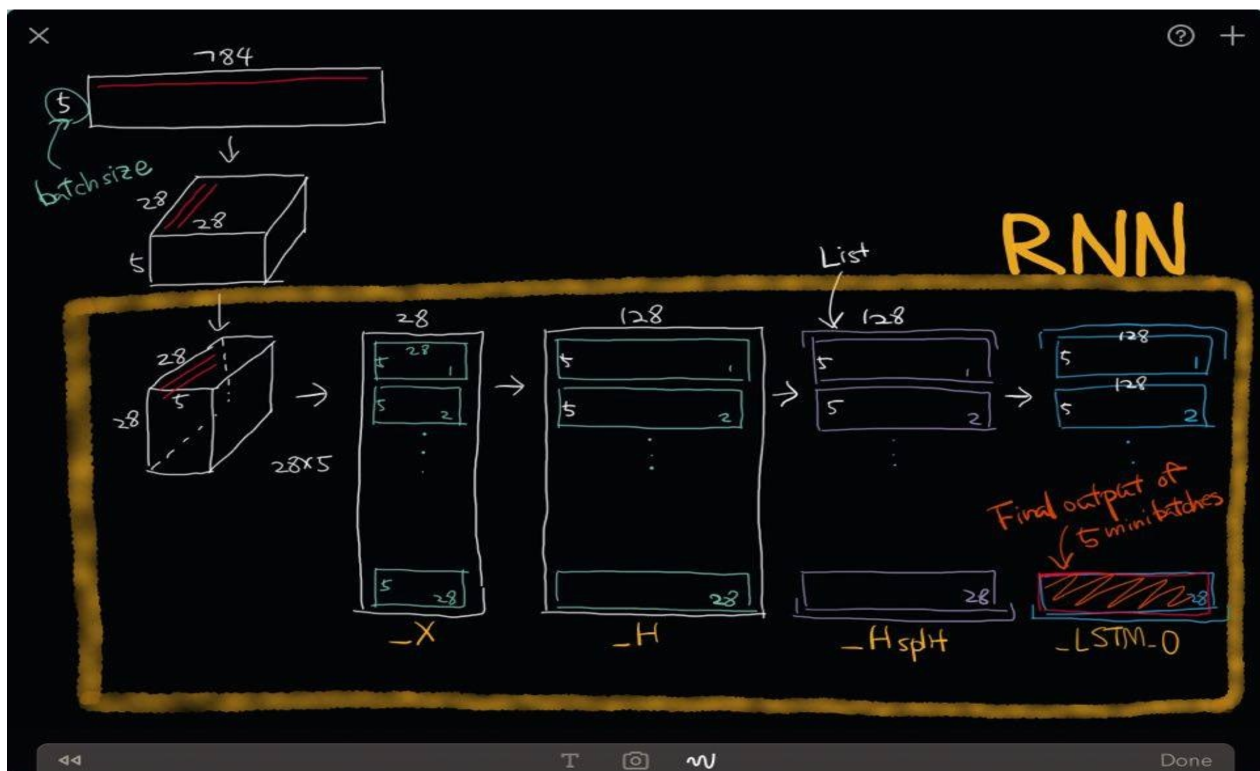
mnist = input_data.read_data_sets("data/", one_hot=True)
trainimgs, trainlabels, testimgs, testlabels \
 = mnist.train.images, mnist.train.labels, mnist.test.images, mn
ist.test.labels
ntrain, ntest, dim, nclasses \
 = trainimgs.shape[0], testimgs.shape[0], trainimgs.shape[1], tr
ainlabels.shape[1]
print ("MNIST loaded")
```

```
Packages imported
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
MNIST loaded
```

We will treat the MNIST image $\in \mathcal{R}^{28 \times 28}$ as 28 sequences of a vector $\mathbf{x} \in \mathcal{R}^{28}$.

Our simple RNN consists of

1. One input layer which converts a 28 dimensional input to an 128 dimensional hidden layer,
2. One intermediate recurrent neural network (LSTM)
3. One output layer which converts an 128 dimensional output of the LSTM to 10 dimensional output indicating a class label.



Contruct a Recurrent Neural Network

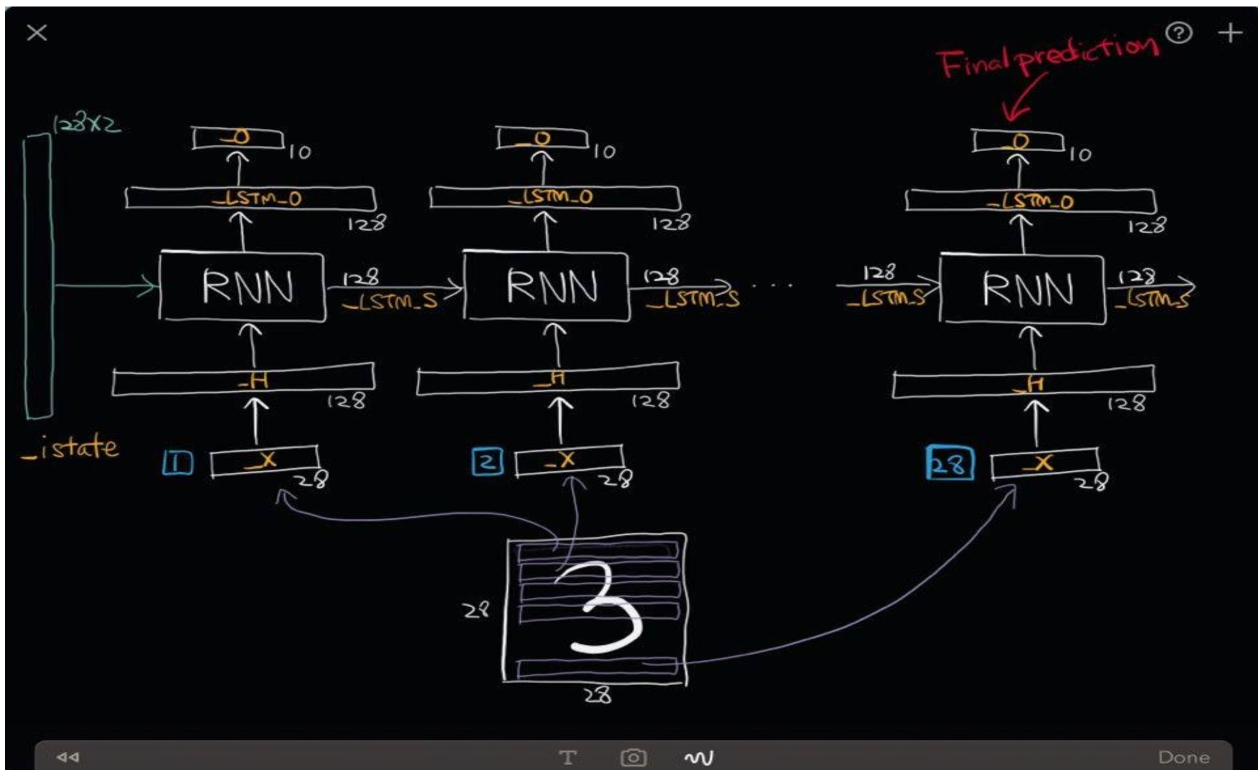
```
diminput  = 28
dimhidden = 128
dimoutput = nclasses
nsteps    = 28
weights = {
    'hidden': tf.Variable(tf.random_normal([diminput, dimhidden]
)),
    'out': tf.Variable(tf.random_normal([dimhidden, dimoutput]))
}
biases = {
    'hidden': tf.Variable(tf.random_normal([dimhidden])),
    'out': tf.Variable(tf.random_normal([dimoutput]))
}
```

```
def _RNN(_X, _istate, _W, _b, _nsteps, _name):
    # 1. Permute input from [batchsize, nsteps, diminput]
    #   => [nsteps, batchsize, diminput]
    _X = tf.transpose(_X, [1, 0, 2])
    # 2. Reshape input to [nsteps*batchsize, diminput]
    _X = tf.reshape(_X, [-1, diminput])
    # 3. Input layer => Hidden layer
    _H = tf.matmul(_X, _W['hidden']) + _b['hidden']
    # 4. Splite data to 'nsteps' chunks. An i-th chunck indicate
s i-th batch data
    _Hsplit = tf.split(0, _nsteps, _H)
    # 5. Get LSTM's final output (_LSTM_O) and state (_LSTM_S)
    #    Both _LSTM_O and _LSTM_S consist of 'batchsize' elements

    #    Only _LSTM_O will be used to predict the output.
    with tf.variable_scope(_name):
        lstm_cell = tf.nn.rnn_cell.BasicLSTMCell(dimhidden, forg
et_bias=1.0)
        _LSTM_O, _LSTM_S = tf.nn.rnn(lstm_cell, _Hsplit, initial
_state=_istate)
    # 6. Output
    _O = tf.matmul(_LSTM_O[-1], _W['out']) + _b['out']
    # Return!
    return {
        'X': _X, 'H': _H, 'Hsplit': _Hsplit,
        'LSTM_O': _LSTM_O, 'LSTM_S': _LSTM_S, 'O': _O
    }
print ("Network ready")
```

```
Network ready
```

Out Network looks like this



Define functions

```
learning_rate = 0.001
x      = tf.placeholder("float", [None, nsteps, diminput])
istate = tf.placeholder("float", [None, 2*dimhidden])
    # state & cell => 2x n_hidden
y      = tf.placeholder("float", [None, dimoutput])
myrnn  = _RNN(x, istate, weights, biases, nsteps, 'basic')
pred   = myrnn['O']
cost   = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
pred, y))
optm   = tf.train.AdamOptimizer(learning_rate).minimize(cost) #
Adam Optimizer
accr   = tf.reduce_mean(tf.cast(tf.equal(tf.argmax(pred,1), tf.a
rgmax(y,1)), tf.float32))
init   = tf.initialize_all_variables()
print ("Network Ready!")
```

```
Network Ready!
```

Run!

```
training_epochs = 5
batch_size      = 128
display_step    = 1
sess = tf.Session()
sess.run(init)
summary_writer = tf.train.SummaryWriter('/tmp/tensorflow_logs',
graph=sess.graph)
print ("Start optimization")
for epoch in range(training_epochs):
    avg_cost = 0.
    total_batch = int(mnist.train.num_examples/batch_size)
    # Loop over all batches
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        batch_xs = batch_xs.reshape((batch_size, nsteps, diminpu
t))
        # Fit training using batch data
        feeds = {x: batch_xs, y: batch_ys, istate: np.zeros((bat
ch_size, 2*dimhidden))}
        sess.run(optm, feed_dict=feeds)
        # Compute average loss
        avg_cost += sess.run(cost, feed_dict=feeds)/total_batch
    # Display logs per epoch step
    if epoch % display_step == 0:
        print ("Epoch: %03d/%03d cost: %.9f" % (epoch, training_
epochs, avg_cost))
        feeds = {x: batch_xs, y: batch_ys, istate: np.zeros((bat
ch_size, 2*dimhidden))}
        train_acc = sess.run(accr, feed_dict=feeds)
        print (" Training accuracy: %.3f" % (train_acc))
        testimgs = testimgs.reshape((ntest, nsteps, diminput))
        feeds = {x: testimgs, y: testlabels, istate: np.zeros((n
test, 2*dimhidden))}
        test_acc = sess.run(accr, feed_dict=feeds)
        print (" Test accuracy: %.3f" % (test_acc))
print ("Optimization Finished.")
```

```
Start optimization
Epoch: 000/005 cost: 0.479400075
 Training accuracy: 0.992
 Test accuracy: 0.922
Epoch: 001/005 cost: 0.136942688
 Training accuracy: 0.969
 Test accuracy: 0.959
Epoch: 002/005 cost: 0.081425477
 Training accuracy: 0.984
 Test accuracy: 0.951
Epoch: 003/005 cost: 0.061170839
 Training accuracy: 0.969
 Test accuracy: 0.973
Epoch: 004/005 cost: 0.047727333
 Training accuracy: 0.992
 Test accuracy: 0.973
Optimization Finished.
```

What we have done so far is to feed 28 sequences of vectors $ \mathbf{x} \in \mathcal{R}^{28}$.

What will happen if we feed first 25 sequences of $\mathbf{x}$?

```
# How may sequences will we use?
nsteps2     = 25

# Test with truncated inputs
testimgs = testimgs.reshape((ntest, nsteps, diminput))
testimgs_trucated = np.zeros(testimgs.shape)
testimgs_trucated[:, 28-nsteps2:] = testimgs[:, :nsteps2, :]
feeds = {x: testimgs_trucated, y: testlabels, istate: np.zeros((
ntest, 2*dimhidden))}
test_acc = sess.run(accr, feed_dict=feeds)
print (" If we use %d seqs, test accuracy becomes %.3f" % (nstep
s2, test_acc))
```

```
 If we use 25 seqs, test accuracy becomes 0.766
```

What's going on inside the RNN?

Inputs to the RNN

```
batch_size = 5
xtest, _ = mnist.test.next_batch(batch_size)
print ("Shape of 'xtest' is %s" % (xtest.shape,))
```

```
Shape of 'xtest' is (5, 784)
```

Reshaped inputs

```
# Reshape (this will go into the network)
xtest1 = xtest.reshape((batch_size, nsteps, diminput))
print ("Shape of 'xtest1' is %s" % (xtest1.shape,))
```

```
Shape of 'xtest1' is (5, 28, 28)
```

Feeds: inputs and initial states

```
feeds = {x: xtest1, istate: np.zeros((batch_size, 2*dimhidden))}
```

Each indivisual input to the LSTM

```
rnnout_X = sess.run(myrnn['X'], feed_dict=feeds)
print ("Shape of 'rnnout_X' is %s" % (rnnout_X.shape,))
```

```
Shape of 'rnnout_X' is (140, 28)
```

Each indivisual intermediate state

```
rnnout_H = sess.run(myrnn['H'], feed_dict=feeds)
print ("Shape of 'rnnout_H' is %s" % (rnnout_H.shape,))
```

```
Shape of 'rnnout_H' is (140, 128)
```

Actual input to the LSTM (List)

```
rnnout_Hsplit = sess.run(myrnn['Hsplit'], feed_dict=feeds)
print ("Type of 'rnnout_Hsplit' is %s" % (type(rnnout_Hsplit)))
print ("Length of 'rnnout_Hsplit' is %s and the shape of each it
em is %s"
       % (len(rnnout_Hsplit), rnnout_Hsplit[0].shape))
```

```
Type of 'rnnout_Hsplit' is <type 'list'>
Length of 'rnnout_Hsplit' is 28 and the shape of each item is (5
, 128)
```

Output from the LSTM (List)

```
rnnout_LSTM_O = sess.run(myrnn['LSTM_O'], feed_dict=feeds)
print ("Type of 'rnnout_LSTM_O' is %s" % (type(rnnout_LSTM_O)))
print ("Length of 'rnnout_LSTM_O' is %s and the shape of each it
em is %s"
       % (len(rnnout_LSTM_O), rnnout_LSTM_O[0].shape))
```

```
Type of 'rnnout_LSTM_O' is <type 'list'>
Length of 'rnnout_LSTM_O' is 28 and the shape of each item is (5
, 128)
```

Final prediction

```
rnnout_O = sess.run(myrnn['O'], feed_dict=feeds)
print ("Shape of 'rnnout_O' is %s" % (rnnout_O.shape,))
```

```
Shape of 'rnnout_O' is (5, 10)
```

```
# Import Packages
import numpy as np
import tensorflow as tf
import collections
import argparse
import time
import os
from six.moves import cPickle
print ("Packages Imported")
```

```
Packages Imported
```

```
# Load text
# data_dir    = "data/tinyshakespeare"
data_dir    = "data/linux_kernel"
save_dir    = "data/linux_kernel"
input_file  = os.path.join(data_dir, "input.txt")
with open(input_file, "r") as f:
    data = f.read()
print ("Text loaded from '%s'" % (input_file))
```

```
Text loaded from 'data/linux_kernel/input.txt'
```

```
# Preprocess Text
# First, count the number of characters
counter = collections.Counter(data)
count_pairs = sorted(counter.items(), key=lambda x: -x[1]) # <=
Sort
print ("Type of 'counter.items()' is %s and length is %d"
       % (type(counter.items()), len(counter.items())))
for i in range(5):
    print ("[%d/%d]" % (i, 3)), # <= This comma remove '\n'
    print (list(counter.items())[i])

print (" ")
print ("Type of 'count_pairs' is %s and length is %d"
       % (type(count_pairs), len(count_pairs)))
for i in range(5):
    print ("[%d/%d]" % (i, 3)), # <= This comma remove '\n'
    print (count_pairs[i])
```

```
Type of 'counter.items()' is <type 'list'> and length is 99
[0/3] (' ', 171222)
[1/3] ('$', 61)
[2/3] ('(', 23412)
[3/3] (',', 17025)
[4/3] ('0', 4322)

Type of 'count_pairs' is <type 'list'> and length is 99
[0/3] (' ', 171222)
[1/3] ('e', 113021)
[2/3] ('t', 102154)
[3/3] ('r', 76185)
[4/3] ('i', 75486)
```

```
# Let's make dictionary
chars, counts = zip(*count_pairs)
vocab = dict(zip(chars, range(len(chars))))
print ("Type of 'chars' is %s and length is %d"
    % (type(chars), len(chars)))
for i in range(5):
    print ("[%d/%d]" % (i, 3)), # <= This comma remove '\n'
    print ("chars[%d] is '%s'" % (i, chars[i]))

print ("")

print ("Type of 'vocab' is %s and length is %d"
    % (type(vocab), len(vocab)))
for i in range(5):
    print ("[%d/%d]" % (i, 3)), # <= This comma remove '\n'
    print ("vocab['%s'] is %s" % (chars[i], vocab[chars[i]]))

# SAve chars and vocab
with open(os.path.join(save_dir, 'chars_vocab.pkl'), 'wb') as f:
    cPickle.dump((chars, vocab), f)
```

```
Type of 'chars' is <type 'tuple'> and length is 99
[0/3] chars[0] is ' '
[1/3] chars[1] is 'e'
[2/3] chars[2] is 't'
[3/3] chars[3] is 'r'
[4/3] chars[4] is 'i'

Type of 'vocab' is <type 'dict'> and length is 99
[0/3] vocab[' '] is 0
[1/3] vocab['e'] is 1
[2/3] vocab['t'] is 2
[3/3] vocab['r'] is 3
[4/3] vocab['i'] is 4
```

chars[0] converts index to char

vocab['a'] converts char to index

```
# Now convert all text to index using vocab!
corpus = np.array(list(map(vocab.get, data)))
print ("Type of 'corpus' is %s, shape is %s, and length is %d"
    % (type(corpus), corpus.shape, len(corpus)))

check_len = 10
print ("\n'corpus' looks like %s" % (corpus[0:check_len]))
for i in range(check_len):
    _wordidx = corpus[i]
    print ("[%d/%d] chars[%02d] corresponds to '%s'"
           % (i, check_len, _wordidx, chars[_wordidx]))
```

```
Type of 'corpus' is <type 'numpy.ndarray'>, shape is (1708871,),
 and length is 1708871

'corpus' looks like [36 22  7  0 22  0  0 13  4  8]
[0/10] chars[36] corresponds to '/'
[1/10] chars[22] corresponds to '*'
[2/10] chars[07] corresponds to '
'
[3/10] chars[00] corresponds to ' '
[4/10] chars[22] corresponds to '*'
[5/10] chars[00] corresponds to ' '
[6/10] chars[00] corresponds to ' '
[7/10] chars[13] corresponds to 'l'
[8/10] chars[04] corresponds to 'i'
[9/10] chars[08] corresponds to 'n'
```

```
# Generate batch data
batch_size  = 50
seq_length  = 200
num_batches = int(corpus.size / (batch_size * seq_length))
# First, reduce the length of corpus to fit batch_size
corpus_reduced = corpus[:(num_batches*batch_size*seq_length)]
xdata = corpus_reduced
ydata = np.copy(xdata)
ydata[:-1] = xdata[1:]
ydata[-1]  = xdata[0]
print ('xdata is ... %s and length is %d' % (xdata, xdata.size))
print ('ydata is ... %s and length is %d' % (ydata, xdata.size))
print ("")

# Second, make batch
xbatches = np.split(xdata.reshape(batch_size, -1), num_batches, 1
)
ybatches = np.split(ydata.reshape(batch_size, -1), num_batches, 1
)
print ("Type of 'xbatches' is %s and length is %d"
    % (type(xbatches), len(xbatches)))
print ("Type of 'ybatches' is %s and length is %d"
    % (type(ybatches), len(ybatches)))
print ("")

# How can we access to xbatches??
nbatch = 5
temp = xbatches[0:nbatch]
print ("Type of 'temp' is %s and length is %d"
    % (type(temp), len(temp)))
for i in range(nbatch):
    temp2 = temp[i]
    print ("Type of 'temp[%d]' is %s and shape is %s" % (i, type
(temp2), temp2.shape,))
```

```
xdata is ... [36 22  7 ..., 11 25  3] and length is 1700000
ydata is ... [22  7  0 ..., 25  3 36] and length is 1700000

Type of 'xbatches' is <type 'list'> and length is 170
Type of 'ybatches' is <type 'list'> and length is 170

Type of 'temp' is <type 'list'> and length is 5
Type of 'temp[0]' is <type 'numpy.ndarray'> and shape is (50, 20
0)
Type of 'temp[1]' is <type 'numpy.ndarray'> and shape is (50, 20
0)
Type of 'temp[2]' is <type 'numpy.ndarray'> and shape is (50, 20
0)
Type of 'temp[3]' is <type 'numpy.ndarray'> and shape is (50, 20
0)
Type of 'temp[4]' is <type 'numpy.ndarray'> and shape is (50, 20
0)
```

Now, we are ready to make our RNN model with seq2seq

```
# Important RNN parameters
vocab_size = len(vocab)
rnn_size   = 128
num_layers = 2
grad_clip  = 5.

# Construct RNN model
unitcell   = tf.nn.rnn_cell.BasicLSTMCell(rnn_size)
cell       = tf.nn.rnn_cell.MultiRNNCell([unitcell] * num_layers
)
input_data = tf.placeholder(tf.int32, [batch_size, seq_length])
targets    = tf.placeholder(tf.int32, [batch_size, seq_length])
istate     = cell.zero_state(batch_size, tf.float32)
# Weigths
with tf.variable_scope('rnnlm'):
    softmax_w = tf.get_variable("softmax_w", [rnn_size, vocab_si
ze])
    softmax_b = tf.get_variable("softmax_b", [vocab_size])
    with tf.device("/cpu:0"):
        embedding = tf.get_variable("embedding", [vocab_size, rn
n_size])
        inputs = tf.split(1, seq_length, tf.nn.embedding_lookup(
embedding, input_data))
        inputs = [tf.squeeze(_input, [1]) for _input in inputs]
# Output
def loop(prev, _):
    prev = tf.nn.xw_plus_b(prev, softmax_w, softmax_b)
```

```
    prev_symbol = tf.stop_gradient(tf.argmax(prev, 1))
    return tf.nn.embedding_lookup(embedding, prev_symbol)
"""
    loop_function: If not None, this function will be applied to
 the i-th output
    in order to generate the i+1-st input, and decoder_inputs wi
ll be ignored,
    except for the first element ("GO" symbol).
"""
outputs, last_state = tf.nn.seq2seq.rnn_decoder(inputs, istate,
cell
                , loop_function=None, scope='rnnlm')
output = tf.reshape(tf.concat(1, outputs), [-1, rnn_size])
logits = tf.nn.xw_plus_b(output, softmax_w, softmax_b)
probs  = tf.nn.softmax(logits)
# Loss
loss = tf.nn.seq2seq.sequence_loss_by_example([logits], # Input
    [tf.reshape(targets, [-1])], # Target
    [tf.ones([batch_size * seq_length])], # Weight
    vocab_size)
# Optimizer
cost     = tf.reduce_sum(loss) / batch_size / seq_length
final_state = last_state
lr       = tf.Variable(0.0, trainable=False)
tvars    = tf.trainable_variables()
grads, _ = tf.clip_by_global_norm(tf.gradients(cost, tvars), gra
d_clip)
_optm    = tf.train.AdamOptimizer(lr)
optm     = _optm.apply_gradients(zip(grads, tvars))

print ("Network Ready")
```

```
Network Ready
```

```
# Train the model!
num_epochs    = 50
save_every    = 500
learning_rate = 0.002
decay_rate    = 0.97

sess = tf.Session()
sess.run(tf.initialize_all_variables())
summary_writer = tf.train.SummaryWriter(save_dir, graph=sess.gra
ph)
saver = tf.train.Saver(tf.all_variables())
init_time = time.time()
for epoch in range(num_epochs):
    # Learning rate scheduling
    sess.run(tf.assign(lr, learning_rate * (decay_rate ** epoch)
))
    state     = sess.run(istate)
    batchidx  = 0
    for iteration in range(num_batches):
        start_time   = time.time()
        randbatchidx = np.random.randint(num_batches)
        xbatch       = xbatches[batchidx]
        ybatch       = ybatches[batchidx]
        batchidx     = batchidx + 1

        # Note that, num_batches = len(xbatches)
        # Train!
        train_loss, state, _ = sess.run([cost, final_state, optm
]
            , feed_dict={input_data: xbatch, targets: ybatch, is
tate: state})
        total_iter = epoch*num_batches + iteration
        end_time     = time.time();
        duration     = end_time - start_time

        if total_iter % 100 == 0:
            print ("[%d/%d] cost: %.4f / Each batch learning too
k %.4f sec"
                   % (total_iter, num_epochs*num_batches, train_
loss, duration))
        if total_iter % save_every == 0:
            ckpt_path = os.path.join(save_dir, 'model.ckpt')
            saver.save(sess, ckpt_path, global_step = total_iter
)
            # Save network!
            print("model saved to '%s'" % (ckpt_path))
```

```
[0/8500] cost: 5.1518 / Each batch learning took 6.2978 sec
model saved to 'data/linux_kernel/model.ckpt'
[100/8500] cost: 3.0705 / Each batch learning took 0.3866 sec
```

```
[200/8500] cost: 2.5382 / Each batch learning took 0.3910 sec
[300/8500] cost: 2.3884 / Each batch learning took 0.5311 sec
[400/8500] cost: 2.2029 / Each batch learning took 0.3930 sec
[500/8500] cost: 1.9560 / Each batch learning took 0.5088 sec
model saved to 'data/linux_kernel/model.ckpt'
[600/8500] cost: 1.9134 / Each batch learning took 0.3861 sec
[700/8500] cost: 1.7579 / Each batch learning took 0.5502 sec
[800/8500] cost: 1.7580 / Each batch learning took 0.4546 sec
[900/8500] cost: 1.6952 / Each batch learning took 0.3958 sec
[1000/8500] cost: 1.5991 / Each batch learning took 0.4516 sec
model saved to 'data/linux_kernel/model.ckpt'
[1100/8500] cost: 1.6036 / Each batch learning took 0.3708 sec
[1200/8500] cost: 1.4374 / Each batch learning took 0.4035 sec
[1300/8500] cost: 1.5513 / Each batch learning took 0.4629 sec
[1400/8500] cost: 1.4814 / Each batch learning took 0.5162 sec
[1500/8500] cost: 1.4986 / Each batch learning took 0.4023 sec
model saved to 'data/linux_kernel/model.ckpt'
[1600/8500] cost: 1.4957 / Each batch learning took 0.5584 sec
[1700/8500] cost: 1.4569 / Each batch learning took 0.5504 sec
[1800/8500] cost: 1.3966 / Each batch learning took 0.4409 sec
[1900/8500] cost: 1.3742 / Each batch learning took 0.8715 sec
[2000/8500] cost: 1.4071 / Each batch learning took 0.7707 sec
model saved to 'data/linux_kernel/model.ckpt'
[2100/8500] cost: 1.4037 / Each batch learning took 0.4636 sec
[2200/8500] cost: 1.3220 / Each batch learning took 0.6967 sec
[2300/8500] cost: 1.3267 / Each batch learning took 0.7644 sec
[2400/8500] cost: 1.2870 / Each batch learning took 0.5228 sec
[2500/8500] cost: 1.3171 / Each batch learning took 0.5671 sec
model saved to 'data/linux_kernel/model.ckpt'
[2600/8500] cost: 1.2876 / Each batch learning took 0.5576 sec
[2700/8500] cost: 1.2571 / Each batch learning took 0.4314 sec
[2800/8500] cost: 1.3123 / Each batch learning took 0.5939 sec
[2900/8500] cost: 1.1588 / Each batch learning took 0.6087 sec
[3000/8500] cost: 1.2834 / Each batch learning took 0.5066 sec
model saved to 'data/linux_kernel/model.ckpt'
[3100/8500] cost: 1.2362 / Each batch learning took 0.4319 sec
[3200/8500] cost: 1.2768 / Each batch learning took 0.4418 sec
[3300/8500] cost: 1.2836 / Each batch learning took 0.6158 sec
[3400/8500] cost: 1.2830 / Each batch learning took 0.7412 sec
[3500/8500] cost: 1.2296 / Each batch learning took 0.7596 sec
model saved to 'data/linux_kernel/model.ckpt'
[3600/8500] cost: 1.2142 / Each batch learning took 0.8046 sec
[3700/8500] cost: 1.2474 / Each batch learning took 0.8149 sec
[3800/8500] cost: 1.2455 / Each batch learning took 0.9514 sec
[3900/8500] cost: 1.1910 / Each batch learning took 1.0230 sec
[4000/8500] cost: 1.1874 / Each batch learning took 0.7037 sec
model saved to 'data/linux_kernel/model.ckpt'
[4100/8500] cost: 1.1602 / Each batch learning took 0.6907 sec
[4200/8500] cost: 1.1896 / Each batch learning took 0.6589 sec
[4300/8500] cost: 1.1680 / Each batch learning took 0.6051 sec
[4400/8500] cost: 1.1472 / Each batch learning took 0.4314 sec
[4500/8500] cost: 1.2073 / Each batch learning took 0.7571 sec
model saved to 'data/linux_kernel/model.ckpt'
```

```
[4600/8500] cost: 1.0601 / Each batch learning took 0.8487 sec
[4700/8500] cost: 1.1822 / Each batch learning took 0.5197 sec
[4800/8500] cost: 1.1427 / Each batch learning took 0.5184 sec
[4900/8500] cost: 1.1774 / Each batch learning took 0.4620 sec
[5000/8500] cost: 1.1902 / Each batch learning took 0.4941 sec
model saved to 'data/linux_kernel/model.ckpt'
[5100/8500] cost: 1.1960 / Each batch learning took 0.7985 sec
[5200/8500] cost: 1.1568 / Each batch learning took 0.7381 sec
[5300/8500] cost: 1.1487 / Each batch learning took 0.5911 sec
[5400/8500] cost: 1.1710 / Each batch learning took 0.8420 sec
[5500/8500] cost: 1.1684 / Each batch learning took 0.7788 sec
model saved to 'data/linux_kernel/model.ckpt'
[5600/8500] cost: 1.1337 / Each batch learning took 0.7290 sec
[5700/8500] cost: 1.1234 / Each batch learning took 1.0153 sec
[5800/8500] cost: 1.1034 / Each batch learning took 0.7469 sec
[5900/8500] cost: 1.1276 / Each batch learning took 0.7259 sec
[6000/8500] cost: 1.1073 / Each batch learning took 0.7722 sec
model saved to 'data/linux_kernel/model.ckpt'
[6100/8500] cost: 1.0955 / Each batch learning took 0.7700 sec
[6200/8500] cost: 1.1489 / Each batch learning took 0.4165 sec
[6300/8500] cost: 1.0120 / Each batch learning took 0.7359 sec
[6400/8500] cost: 1.1296 / Each batch learning took 0.6871 sec
[6500/8500] cost: 1.0963 / Each batch learning took 0.6530 sec
model saved to 'data/linux_kernel/model.ckpt'
[6600/8500] cost: 1.1259 / Each batch learning took 0.4506 sec
[6700/8500] cost: 1.1422 / Each batch learning took 0.3957 sec
[6800/8500] cost: 1.1431 / Each batch learning took 0.4530 sec
[6900/8500] cost: 1.1168 / Each batch learning took 0.4068 sec
[7000/8500] cost: 1.1119 / Each batch learning took 1.0343 sec
model saved to 'data/linux_kernel/model.ckpt'
[7100/8500] cost: 1.1255 / Each batch learning took 0.4080 sec
[7200/8500] cost: 1.1266 / Each batch learning took 0.3840 sec
[7300/8500] cost: 1.1036 / Each batch learning took 0.8628 sec
[7400/8500] cost: 1.0860 / Each batch learning took 0.4150 sec
[7500/8500] cost: 1.0681 / Each batch learning took 0.4738 sec
model saved to 'data/linux_kernel/model.ckpt'
[7600/8500] cost: 1.0921 / Each batch learning took 0.4141 sec
[7700/8500] cost: 1.0728 / Each batch learning took 0.3944 sec
[7800/8500] cost: 1.0644 / Each batch learning took 0.4473 sec
[7900/8500] cost: 1.1155 / Each batch learning took 0.4841 sec
[8000/8500] cost: 0.9819 / Each batch learning took 0.4198 sec
model saved to 'data/linux_kernel/model.ckpt'
[8100/8500] cost: 1.0945 / Each batch learning took 0.4452 sec
[8200/8500] cost: 1.0682 / Each batch learning took 0.4038 sec
[8300/8500] cost: 1.0939 / Each batch learning took 0.4889 sec
[8400/8500] cost: 1.1111 / Each batch learning took 0.3995 sec
```

Run the command line

tensorboard --logdir=/tmp/tf_logs/char_rnn_tutorial

Open http://localhost:6006/ into your web browser

```
print ("Done!! It took %.4f second. " %(time.time() - init_time)
)
```

```
Done!! It took 5238.4040 second.
```

```
# Import Packages
import numpy as np
import tensorflow as tf
import collections
import argparse
import time
import os
from six.moves import cPickle
print ("Packages Imported")
```

```
Packages Imported
```

```
# Load chars and vocab
load_dir = "data/linux_kernel"
with open(os.path.join(load_dir, 'chars_vocab.pkl'), 'rb') as f:
    chars, vocab = cPickle.load(f)
vocab_size = len(vocab)
print ("'vocab_size' is %d" % (vocab_size))
```

```
'vocab_size' is 99
```

Now, we are ready to make our RNN model with seq2seq

This network is for sampling, so we don't need batches for sequenes nor optimizers

```
# Important RNN parameters
rnn_size   = 128
num_layers = 2

batch_size = 1 # <= In the training phase, these were both 50
seq_length = 1

# Construct RNN model
unitcell   = tf.nn.rnn_cell.BasicLSTMCell(rnn_size)
cell       = tf.nn.rnn_cell.MultiRNNCell([unitcell] * num_layers
)
input_data = tf.placeholder(tf.int32, [batch_size, seq_length])
istate     = cell.zero_state(batch_size, tf.float32)
# Weigths
with tf.variable_scope('rnnlm'):
    softmax_w = tf.get_variable("softmax_w", [rnn_size, vocab_si
ze])
    softmax_b = tf.get_variable("softmax_b", [vocab_size])
    with tf.device("/cpu:0"):
        embedding = tf.get_variable("embedding", [vocab_size, rn
n_size])
        inputs = tf.split(1, seq_length, tf.nn.embedding_lookup(
embedding, input_data))
        inputs = [tf.squeeze(_input, [1]) for _input in inputs]
# Output
def loop(prev, _):
    prev = tf.nn.xw_plus_b(prev, softmax_w, softmax_b)
    prev_symbol = tf.stop_gradient(tf.argmax(prev, 1))
    return tf.nn.embedding_lookup(embedding, prev_symbol)
outputs, final_state = seq2seq.rnn_decoder(inputs, istate, cell
                                          , loop_function=None,
scope='rnnlm')
output = tf.reshape(tf.concat(1, outputs), [-1, rnn_size])

logits = tf.nn.xw_plus_b(output, softmax_w, softmax_b)
probs  = tf.nn.softmax(logits)

print ("Network Ready")
```

```
Network Ready
```

```
# Restore RNN
sess = tf.Session()
sess.run(tf.initialize_all_variables())
saver = tf.train.Saver(tf.all_variables())
ckpt  = tf.train.get_checkpoint_state(load_dir)

print (ckpt.model_checkpoint_path)
saver.restore(sess, ckpt.model_checkpoint_path)
```

```
data/linux_kernel/model.ckpt-8000
```

Finally, show what RNN has generated!

```
# Sampling function
def weighted_pick(weights):
    t = np.cumsum(weights)
    s = np.sum(weights)
    return(int(np.searchsorted(t, np.random.rand(1)*s)))

# Sample using RNN and prime characters
prime = "/* "
state = sess.run(cell.zero_state(1, tf.float32))
for char in prime[:-1]:
    x = np.zeros((1, 1))
    x[0, 0] = vocab[char]
    state = sess.run(final_state, feed_dict={input_data: x, istate:state})

# Sample 'num' characters
ret  = prime
char = prime[-1] # <= This goes IN!
num  = 1000
for n in range(num):
    x = np.zeros((1, 1))
    x[0, 0] = vocab[char]
    [probsval, state] = sess.run([probs, final_state]
        , feed_dict={input_data: x, istate:state})
    p      = probsval[0]

    sample = weighted_pick(p)
    # sample = np.argmax(p)

    pred   = chars[sample]
    ret    = ret + pred
    char   = pred

print ("Sampling Done. \n___________________________________________\n")

print (ret)
```

```
Sampling Done.
___________________________________________

/*  :  A C. Fruemptly etweennars must be serversed */
static int __cgroup_hash_power(struct rt_mutex_d *uaddr, int watab, long
-XIT_PYS__AUTIMER_PAT(seed_class_table_watch, v1->curr);
}

static void down_cpusets(struct pid;
static int pid_thread(voids_mm)
{
```

```
    if (ps->cpumainte_to_cgroup_grp <= NULL)
        return 0;
}

conset sched_VRICE_SOFTIRQ_DISU{
    softirq_signal(this_css_set_bytes));
}

void private = {
        .mode        = CPUCLOCK_BALANCE,
        .process        = optime)

/*
 * The are
 *    en
 * @buf' - for so allows the condext it of it regions)
 * massessiging that   Sto be stime in the expoxes
 */
void __fsix;
    struct audit_chunk *tsk;

    key_utvec_oper(struct *read_ns, struct futex_ckernel);
    int atomic_attime = res->init_switch(void),
                -+signal->state = 0;
    tmr = tmp;
    printk("%s\n", signal, &max_huts_string, 1, look_t *)(modema
sk++);
    up_sem(cft, &(max))) {
        if (probes)
            set_cpu(name == 0)
            goto out;
    }

    pposs_unlock(*pefmask_plocks);
    audit_log_lock_fuces(rq);
}

static void again;

int
con
```

Hope, it was good.

TRAIN HANGUL-RNN

```
# -*- coding: utf-8 -*-
# Import Packages
import numpy as np
import tensorflow as tf
import collections
import argparse
import time
import os
from six.moves import cPickle
from TextLoader import *
from Hangulpy import *
print ("Packages Imported")
```

```
Packages Imported
```

LOAD DATASET WITH TEXTLOADER

```
data_dir    = "data/nine_dreams"
batch_size  = 50
seq_length  = 50
data_loader = TextLoader(data_dir, batch_size, seq_length)
# This makes "vocab.pkl" and "data.npy" in "data/nine_dreams"
#  from "data/nine_dreams/input.txt"
```

```
loading preprocessed files
```

VOCAB AND CHARS

```
vocab_size = data_loader.vocab_size
vocab = data_loader.vocab
chars = data_loader.chars
print ( "type of 'data_loader.vocab' is %s, length is %d"
       % (type(data_loader.vocab), len(data_loader.vocab)) )
print ( "type of 'data_loader.chars' is %s, length is %d"
       % (type(data_loader.chars), len(data_loader.chars)) )
```

```
type of 'data_loader.vocab' is <type 'dict'>, length is 76
type of 'data_loader.chars' is <type 'tuple'>, length is 76
```

VOCAB: DICTIONARY (CHAR->INDEX)

```
print (data_loader.vocab)
```

```
{u'_': 69, u'6': 59, u':': 57, u'\n': 19, u'4': 67, u'5': 63, u'
>': 75, u'!': 52, u' ': 1, u'"': 28, u'\u1d25': 0, u"'": 49, u')
': 46, u'(': 45, u'-': 65, u',': 27, u'.': 24, u'\u3131': 7, u'0
': 73, u'\u3133': 60, u'\u3132': 29, u'\u3135': 50, u'\u3134': 4
, u'\u3137': 13, u'\u3136': 44, u'\u3139': 5, u'\u3138': 32, u'\
u313b': 55, u'\u313a': 48, u'\u313c': 54, u'?': 41, u'3': 66, u'
\u3141': 12, u'\u3140': 51, u'\u3143': 47, u'\u3142': 17, u'\u31
45': 10, u'\u3144': 43, u'\u3147': 2, u'\u3146': 22, u'\u3149':
40, u'\u3148': 15, u'\u314b': 42, u'\u314a': 23, u'\u314d': 31,
u'\u314c': 30, u'\u314f': 3, u'\u314e': 14, u'\u3151': 34, u'\u3
150': 21, u'\u3153': 11, u'\u3152': 74, u'\u3155': 18, u'\u3154'
: 20, u'\u3157': 9, u'\u3156': 39, u'\u3159': 53, u'\u3158': 26,
 u'\u315b': 38, u'\u315a': 33, u'\u315d': 36, u'\u315c': 16, u'\
u315f': 35, u'\u315e': 61, u'\u3161': 8, u'\u3160': 37, u'\u3163
': 6, u'\u3162': 25, u'\x1a': 72, u'9': 64, u'7': 71, u'2': 62,
u'1': 58, u'\u313f': 56, u'\u313e': 70, u'8': 68}
```

CHARS: LIST (INDEX->CHAR)

```
print (data_loader.chars)
# USAGE
print (data_loader.chars[0])
```

```
(u'\u1d25', u' ', u'\u3147', u'\u314f', u'\u3134', u'\u3139', u'
\u3163', u'\u3131', u'\u3161', u'\u3157', u'\u3145', u'\u3153',
u'\u3141', u'\u3137', u'\u314e', u'\u3148', u'\u315c', u'\u3142'
, u'\u3155', u'\n', u'\u3154', u'\u3150', u'\u3146', u'\u314a',
u'.', u'\u3162', u'\u3158', u',', u'"', u'\u3132', u'\u314c', u'
\u314d', u'\u3138', u'\u315a', u'\u3151', u'\u315f', u'\u315d',
u'\u3160', u'\u315b', u'\u3156', u'\u3149', u'?', u'\u314b', u'\
u3144', u'\u3136', u'(', u')', u'\u3143', u'\u313a', u"'", u'\u3
135', u'\u3140', u'!', u'\u3159', u'\u313c', u'\u313b', u'\u313f
', u':', u'1', u'6', u'\u3133', u'\u315e', u'2', u'5', u'9', u'-
', u'3', u'4', u'8', u'_', u'\u313e', u'7', u'\x1a', u'0', u'\u3
152', u'>')
ᴥ
```

TRAINING BATCH (IMPORTANT!!)

```
x, y = data_loader.next_batch()
print ("Type of 'x' is %s. Shape is %s" % (type(x), x.shape,))
print ("x looks like \n%s" % (x))
print
print ("Type of 'y' is %s. Shape is %s" % (type(y), y.shape,))
print ("y looks like \n%s" % (y))
```

```
Type of 'x' is <type 'numpy.ndarray'>. Shape is (50, 50)
x looks like
[[ 3  5  0 ...,  3  4  0]
 [20  0  1 ..., 13  3  0]
 [10 11  2 ...,  1  7  3]
 ...,
 [ 1 17  6 ...,  0  1  7]
 [ 0 14  3 ..., 12  3  4]
 [ 0  7  3 ...,  1 15  3]]

Type of 'y' is <type 'numpy.ndarray'>. Shape is (50, 50)
y looks like
[[ 5  0  1 ...,  4  0 15]
 [ 0  1  7 ...,  3  0 24]
 [11  2  0 ...,  7  3  0]
 ...,
 [17  6  0 ...,  1  7  9]
 [14  3  0 ...,  3  4  0]
 [ 7  3  0 ..., 15  3  2]]
```

DEFINE A MULTILAYER LSTM NETWORK

```
rnn_size   = 512
num_layers = 3
grad_clip  = 5. # <= GRADIENT CLIPPING (PRACTICALLY IMPORTANT)
vocab_size = data_loader.vocab_size

# SELECT RNN CELL (MULTI LAYER LSTM)
unitcell = tf.nn.rnn_cell.BasicLSTMCell(rnn_size)
cell = tf.nn.rnn_cell.MultiRNNCell([unitcell] * num_layers)

# Set paths to the graph
input_data = tf.placeholder(tf.int32, [batch_size, seq_length])
targets    = tf.placeholder(tf.int32, [batch_size, seq_length])
initial_state = cell.zero_state(batch_size, tf.float32)

# Set Network
with tf.variable_scope('rnnlm'):
    softmax_w = tf.get_variable("softmax_w", [rnn_size, vocab_si
ze])
    softmax_b = tf.get_variable("softmax_b", [vocab_size])
    with tf.device("/cpu:0"):
        embedding = tf.get_variable("embedding", [vocab_size, rn
n_size])
        inputs = tf.split(1, seq_length, tf.nn.embedding_lookup(
                embedding, input_data))
        inputs = [tf.squeeze(input_, [1]) for input_ in inputs]
print ("Network ready")
```

```
Network ready
```

Define functions

```
# Output of RNN
outputs, last_state = tf.nn.seq2seq.rnn_decoder(inputs, initial_
state
                        , cell, loop_function=None, scope='rnnlm'
)
output = tf.reshape(tf.concat(1, outputs), [-1, rnn_size])
logits = tf.nn.xw_plus_b(output, softmax_w, softmax_b)

# Next word probability
probs = tf.nn.softmax(logits)
print ("FUNCTIONS READY")
```

```
FUNCTIONS READY
```

DEFINE LOSS FUNCTION

```
loss = tf.nn.seq2seq.sequence_loss_by_example([logits], # Input
    [tf.reshape(targets, [-1])], # Target
    [tf.ones([batch_size * seq_length])], # Weight
    vocab_size)
print ("LOSS FUNCTION")
```

```
LOSS FUNCTION
```

DEFINE COST FUNCTION

```
cost = tf.reduce_sum(loss) / batch_size / seq_length

# GRADIENT CLIPPING !
lr = tf.Variable(0.0, trainable=False) # <= LEARNING RATE
tvars = tf.trainable_variables()
grads, _ = tf.clip_by_global_norm(tf.gradients(cost, tvars), gra
d_clip)
_optm = tf.train.AdamOptimizer(lr)
optm = _optm.apply_gradients(zip(grads, tvars))

final_state = last_state
print ("NETWORK READY")
```

```
NETWORK READY
```

OPTIMIZE NETWORK WITH LR SCHEDULING

```
num_epochs    = 500
save_every    = 1000
learning_rate = 0.0002
decay_rate    = 0.97

save_dir = 'data/nine_dreams'
sess = tf.Session()
sess.run(tf.initialize_all_variables())
summary_writer = tf.train.SummaryWriter(save_dir
                    , graph=sess.graph)
saver = tf.train.Saver(tf.all_variables())
for e in range(num_epochs): # for all epochs

    # LEARNING RATE SCHEDULING
    sess.run(tf.assign(lr, learning_rate * (decay_rate ** e)))

    data_loader.reset_batch_pointer()
    state = sess.run(initial_state)
    for b in range(data_loader.num_batches):
        start = time.time()
        x, y = data_loader.next_batch()
        feed = {input_data: x, targets: y, initial_state: state}
        # Train!
        train_loss, state, _ = sess.run([cost, final_state, optm
], feed)
        end = time.time()
        # PRINT
        if b % 100 == 0:
            print ("%d/%d (epoch: %d), loss: %.3f, time/batch: %
.3f"
                   % (e * data_loader.num_batches + b
                    , num_epochs * data_loader.num_batches
                    , e, train_loss, end - start))
        # SAVE MODEL
        if (e * data_loader.num_batches + b) % save_every == 0:
            checkpoint_path = os.path.join(save_dir, 'model.ckpt'
)
            saver.save(sess, checkpoint_path
                       , global_step = e * data_loader.num_batch
es + b)
            print("model saved to {}".format(checkpoint_path))
```

```
# IT TAKE A LOOOOOOOOT OF TIME
```

Sample Hangul RNN

```
# -*- coding: utf-8 -*-
# Import Packages
import numpy as np
import tensorflow as tf
import collections
import string
import argparse
import time
import os
from six.moves import cPickle
from TextLoader import *
from Hangulpy import *
print ("Packages Imported")
```

```
Packages Imported
```

Load dataset using TextLoader

```
data_dir    = "data/nine_dreams"
batch_size  = 50
seq_length  = 50
data_loader = TextLoader(data_dir, batch_size, seq_length)
# This makes "vocab.pkl" and "data.npy" in "data/nine_dreams"
#  from "data/nine_dreams/input.txt"
vocab_size = data_loader.vocab_size
vocab = data_loader.vocab
chars = data_loader.chars
print ( "type of 'data_loader' is %s, length is %d"
       % (type(data_loader.vocab), len(data_loader.vocab)) )
print ( "\n" )
print ("data_loader.vocab looks like \n%s " %
       (data_loader.vocab))
print ( "\n" )
print ( "type of 'data_loader.chars' is %s, length is %d"
       % (type(data_loader.chars), len(data_loader.chars)) )
print ( "\n" )
print ("data_loader.chars looks like \n%s " % (data_loader.chars
,))
```

```
loading preprocessed files
type of 'data_loader' is <type 'dict'>, length is 76


data_loader.vocab looks like
{u'_': 69, u'6': 59, u':': 57, u'\n': 19, u'4': 67, u'5': 63, u'
>': 75, u'!': 52, u' ': 1, u'"': 28, u'\u1d25': 0, u"'": 49, u')
': 46, u'(': 45, u'-': 65, u',': 27, u'.': 24, u'\u3131': 7, u'0
': 73, u'\u3133': 60, u'\u3132': 29, u'\u3135': 50, u'\u3134': 4
, u'\u3137': 13, u'\u3136': 44, u'\u3139': 5, u'\u3138': 32, u'\
u313b': 55, u'\u313a': 48, u'\u313c': 54, u'?': 41, u'3': 66, u'
\u3141': 12, u'\u3140': 51, u'\u3143': 47, u'\u3142': 17, u'\u31
45': 10, u'\u3144': 43, u'\u3147': 2, u'\u3146': 22, u'\u3149':
40, u'\u3148': 15, u'\u314b': 42, u'\u314a': 23, u'\u314d': 31,
u'\u314c': 30, u'\u314f': 3, u'\u314e': 14, u'\u3151': 34, u'\u3
150': 21, u'\u3153': 11, u'\u3152': 74, u'\u3155': 18, u'\u3154'
: 20, u'\u3157': 9, u'\u3156': 39, u'\u3159': 53, u'\u3158': 26,
 u'\u315b': 38, u'\u315a': 33, u'\u315d': 36, u'\u315c': 16, u'\
u315f': 35, u'\u315e': 61, u'\u3161': 8, u'\u3160': 37, u'\u3163
': 6, u'\u3162': 25, u'\x1a': 72, u'9': 64, u'7': 71, u'2': 62,
u'1': 58, u'\u313f': 56, u'\u313e': 70, u'8': 68}


type of 'data_loader.chars' is <type 'tuple'>, length is 76


data_loader.chars looks like
(u'\u1d25', u' ', u'\u3147', u'\u314f', u'\u3134', u'\u3139', u'
\u3163', u'\u3131', u'\u3161', u'\u3157', u'\u3145', u'\u3153',
u'\u3141', u'\u3137', u'\u314e', u'\u3148', u'\u315c', u'\u3142'
, u'\u3155', u'\n', u'\u3154', u'\u3150', u'\u3146', u'\u314a',
u'.', u'\u3162', u'\u3158', u',', u'"', u'\u3132', u'\u314c', u'
\u314d', u'\u3138', u'\u315a', u'\u3151', u'\u315f', u'\u315d',
u'\u3160', u'\u315b', u'\u3156', u'\u3149', u'?', u'\u314b', u'\
u3144', u'\u3136', u'(', u')', u'\u3143', u'\u313a', u"'", u'\u3
135', u'\u3140', u'!', u'\u3159', u'\u313c', u'\u313b', u'\u313f
', u':', u'1', u'6', u'\u3133', u'\u315e', u'2', u'5', u'9', u'-
', u'3', u'4', u'8', u'_', u'\u313e', u'7', u'\x1a', u'0', u'\u3
152', u'>')
```

Define Network

```
rnn_size   = 512
num_layers = 3
grad_clip  = 5.

_batch_size = 1
_seq_length = 1
```

```
vocab_size = data_loader.vocab_size

with tf.device("/cpu:0"):
    # Select RNN Cell
    unitcell = tf.nn.rnn_cell.BasicLSTMCell(rnn_size)
    cell = tf.nn.rnn_cell.MultiRNNCell([unitcell] * num_layers)
    # Set paths to the graph
    input_data = tf.placeholder(tf.int32, [_batch_size, _seq_len
gth])
    targets    = tf.placeholder(tf.int32, [_batch_size, _seq_len
gth])
    initial_state = cell.zero_state(_batch_size, tf.float32)

    # Set Network
    with tf.variable_scope('rnnlm'):
        softmax_w = tf.get_variable("softmax_w", [rnn_size, voca
b_size])
        softmax_b = tf.get_variable("softmax_b", [vocab_size])
        with tf.device("/cpu:0"):
            embedding = tf.get_variable("embedding", [vocab_size
, rnn_size])
            inputs = tf.split(1, _seq_length, tf.nn.embedding_lo
okup(embedding, input_data))
            inputs = [tf.squeeze(input_, [1]) for input_ in inpu
ts]

    # Loop function for seq2seq
    def loop(prev, _):
        prev = tf.nn.xw_plus_b(prev, softmax_w, softmax_b)
        prev_symbol = tf.stop_gradient(tf.argmax(prev, 1))
        return tf.nn.embedding_lookup(embedding, prev_symbol)
    # Output of RNN
    outputs, last_state = tf.nn.seq2seq.rnn_decoder(inputs, init
ial_state
                                , cell, loop_function=None, scop
e='rnnlm')
    output = tf.reshape(tf.concat(1, outputs), [-1, rnn_size])
    logits = tf.nn.xw_plus_b(output, softmax_w, softmax_b)
    # Next word probability
    probs = tf.nn.softmax(logits)
    # Define LOSS
    loss = tf.nn.seq2seq.sequence_loss_by_example([logits], # In
put
        [tf.reshape(targets, [-1])], # Target
        [tf.ones([_batch_size * _seq_length])], # Weight
        vocab_size)
    # Define Optimizer
    cost = tf.reduce_sum(loss) / _batch_size / _seq_length
    final_state = last_state
    lr = tf.Variable(0.0, trainable=False)
    tvars = tf.trainable_variables()
    grads, _ = tf.clip_by_global_norm(tf.gradients(cost, tvars),
 grad_clip)
```

```
    _optm = tf.train.AdamOptimizer(lr)
    optm = _optm.apply_gradients(zip(grads, tvars))

print ("Network Ready")
```

```
Network Ready
```

```
# Sample !
def sample( sess, chars, vocab, __probs, num=200, prime=u'ㅇㅗㄴㅡㄹ '):
    state = sess.run(cell.zero_state(1, tf.float32))
    _probs = __probs
    prime = list(prime)
    for char in prime[:-1]:
        x = np.zeros((1, 1))
        x[0, 0] = vocab[char]
        feed = {input_data: x, initial_state:state}
        [state] = sess.run([final_state], feed)

    def weighted_pick(weights):
        weights = weights / np.sum(weights)
        t = np.cumsum(weights)
        s = np.sum(weights)
        return(int(np.searchsorted(t, np.random.rand(1)*s)))

    ret = prime
    char = prime[-1]
    for n in range(num):
        x = np.zeros((1, 1))
        x[0, 0] = vocab[char]
        feed = {input_data: x, initial_state:state}
        [_probsval, state] = sess.run([_probs, final_state], fee
d)
        p = _probsval[0]
        sample = int(np.random.choice(len(p), p=p))
        # sample = weighted_pick(p)
        # sample = np.argmax(p)
        pred = chars[sample]
        ret += pred
        char = pred
    return ret
print ("sampling function done.")
```

```
sampling function done.
```

Sample

```
save_dir = 'data/nine_dreams'
prime = decompose_text(u"누구 ")

print ("Prime Text : %s => %s" % (automata(prime), "".join(prime
)))
n = 2000

sess = tf.Session()
sess.run(tf.initialize_all_variables())
saver = tf.train.Saver(tf.all_variables())
ckpt = tf.train.get_checkpoint_state(save_dir)

# load_name = u'data/nine_dreams/model.ckpt-0'
load_name = u'data/nine_dreams/model.ckpt-99000'

print (load_name)

if ckpt and ckpt.model_checkpoint_path:
    saver.restore(sess, load_name)
    sampled_text = sample(sess, chars, vocab, probs, n, prime)
    #print ("")
    print (u"SAMPLED TEXT = %s" % sampled_text)
    print ("")
    print ("-- RESULT --")
    print (automata("".join(sampled_text)))
```

```
Prime Text : 누구  => ㄴㅜᴥㄱㅜᴥ
data/nine_dreams/model.ckpt-99000
SAMPLED TEXT = [u'\u3134', u'\u315c', u'\u1d25', u'\u3131', u'\u
315c', u'\u1d25', u' ', u'\u3145', u'\u3157', u'\u1d25', u'\u313
9', u'\u3163', u'\u1d25', u'\u3147', u'\u3154', u'\u1d25', u' ',
 u'\u3145', u'\u3153', u'\u1d25', u' ', u'\u3147', u'\u3163', u'
\u3146', u'\u1d25', u'\u3147', u'\u3161', u'\u3139', u'\u1d25',
u' ', u'\u3131', u'\u3153', u'\u3145', u'\u1d25', u'\u3147', u'\
u3163', u'\u1d25', u'\u3139', u'\u3157', u'\u1d25', u'\u3137', u
'\u314f', u'\u1d25', u'.', u'"', u'\n', u' ', u' ', u'\u3147', u
'\u3163', u'\u1d25', u'\u3147', u'\u3153', u'\u1d25', u'\u3145',
 u'\u3153', u'\u1d25', u' ', u'\u3145', u'\u3153', u'\u1d25', u'
\u3147', u'\u315c', u'\u3139', u'\u1d25', u'\u3147', u'\u3154',
u'\u1d25', u' ', u'\u3137', u'\u3161', u'\u3139', u'\u1d25', u'\
u3147', u'\u3153', u'\u1d25', u'\u3131', u'\u314f', u'\u1d25', u
'\u3134', u'\u3163', u'\u1d25', u' ', u'\u3147', u'\u3153', u'\u
1d25', u'\u3149', u'\u3163', u'\u1d25', u' ', u'\u3147', u'\u316
3', u'\u1d25', u' ', u'\u3141', u'\u314f', u'\u3139', u'\u1d25',
 u'\u3147', u'\u3161', u'\u3139', u'\u1d25', u' ', u'\u3137', u'
\u3161', u'\u3137', u'\u1d25', u'\u3131', u'\u3157', u'\u1d25',
u' ', u'\u3147', u'\u3163', u'\u3146', u'\u1d25', u'\u3147', u'\
```

```
u3161', u'\u1d25', u'\u3134', u'\u3163', u'\u1d25', u' ', u'\u31
47', u'\u3163', u'\u1d25', u' ', u'\u3141', u'\u314f', u'\u3139'
, u'\u1d25', u'\u3147', u'\u3161', u'\u3139', u'\u1d25', u' ', u
'\u3137', u'\u3161', u'\u3137', u'\u1d25', u'\u3131', u'\u3157',
 u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u3146', u'\u1d25', u'
\u3147', u'\u3161', u'\u1d25', u'\u3134', u'\u3163', u'\u1d25',
u' ', u'\u3147', u'\u3163', u'\u1d25', u' ', u'\u3141', u'\u314f
', u'\u3139', u'\u1d25', u'\u3147', u'\u3161', u'\u3139', u'\u1d
25', u' ', u'\u3137', u'\u3161', u'\u3137', u'\u1d25', u'\u3131'
, u'\u3157', u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u3146', u
'\u1d25', u'\u3147', u'\u3153', u'\u3146', u'\u1d25', u'\u3137',
 u'\u314f', u'\u1d25', u'.', u' ', u'\n', u' ', u' ', u'\u3147',
 u'\u3163', u'\u1d25', u'\u3147', u'\u3154', u'\u1d25', u' ', u'
\u3137', u'\u3150', u'\u1d25', u'\u3137', u'\u314f', u'\u3142',
u'\u1d25', u'\u314e', u'\u314f', u'\u1d25', u'\u3131', u'\u3163'
, u'\u1d25', u'\u3139', u'\u3161', u'\u3139', u'\u1d25', u',', u
' ', u'\n', u' ', u' ', u'"', u'\u3145', u'\u3157', u'\u1d25', u
'\u3147', u'\u3160', u'\u1d25', u'\u3131', u'\u314f', u'\u1d25',
 u' ', u'\u3137', u'\u314f', u'\u1d25', u'\u3145', u'\u3163', u'
\u1d25', u' ', u'\u3147', u'\u3163', u'\u1d25', u'\u3139', u'\u3
161', u'\u1d25', u'\u3131', u'\u3163', u'\u1d25', u'\u3139', u'\
u3161', u'\u3139', u'\u1d25', u',', u' ', u'\n', u' ', u' ', u'"
', u'\u3147', u'\u3163', u'\u1d25', u'\u3148', u'\u3154', u'\u1d
25', u' ', u'\u3147', u'\u3163', u'\u1d25', u'\u3147', u'\u3154'
, u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u1d25', u'\u3139', u
'\u3161', u'\u1d25', u'\u3131', u'\u3163', u'\u1d25', u'\u3139',
 u'\u3161', u'\u3139', u'\u1d25', u',', u' ', u'\n', u' ', u' ',
 u'"', u'\u3145', u'\u3157', u'\u1d25', u'\u3148', u'\u3153', u'
\u1d25', u'\u3131', u'\u314f', u'\u1d25', u' ', u'\u3147', u'\u3
153', u'\u1d25', u'\u3149', u'\u3163', u'\u1d25', u' ', u'\u3147
', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u1d25', u'\u31
31', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u3139', u'\u
1d25', u',', u' ', u'\n', u' ', u' ', u'"', u'\u3145', u'\u3157'
, u'\u1d25', u'\u3148', u'\u3153', u'\u1d25', u'\u3134', u'\u316
1', u'\u3134', u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u1d25',
 u' ', u'\u3141', u'\u314f', u'\u3139', u'\u1d25', u'\u3147', u'
\u3161', u'\u3139', u'\u1d25', u' ', u'\u3137', u'\u3161', u'\u3
137', u'\u1d25', u'\u3131', u'\u3157', u'\u1d25', u' ', u'\u3147
', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u1d25', u'\u31
31', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u3139', u'\u
1d25', u',', u' ', u'\n', u' ', u' ', u'"', u'\u3145', u'\u3157'
, u'\u1d25', u'\u3148', u'\u3153', u'\u1d25', u'\u3131', u'\u314
f', u'\u1d25', u' ', u'\u3147', u'\u3153', u'\u1d25', u'\u3149',
 u'\u3163', u'\u1d25', u' ', u'\u3145', u'\u314f', u'\u1d25', u'
\u3139', u'\u314f', u'\u3141', u'\u1d25', u'\u3147', u'\u3161',
u'\u3139', u'\u1d25', u' ', u'\u3137', u'\u3161', u'\u3139', u'\
u1d25', u'\u3147', u'\u3153', u'\u1d25', u' ', u'\u3131', u'\u31
4f', u'\u1d25', u'\u3145', u'\u3153', u'\u1d25', u' ', u'\u3131'
, u'\u3161', u'\u1d25', u' ', u'\u3147', u'\u314f', u'\u1d25', u
'\u3139', u'\u3161', u'\u3141', u'\u1d25', u'\u3137', u'\u314f',
 u'\u1d25', u'\u3147', u'\u315c', u'\u3134', u'\u1d25', u' ', u'
\u3147', u'\u3163', u'\u3139', u'\u1d25', u'\u3147', u'\u3163',
u'\u1d25', u'\u3147', u'\u3153', u'\u3146', u'\u1d25', u'\u3137'
```

```
, u'\u314f', u'\u1d25', u'.', u' ', u'\n', u' ', u' ', u'\u3147'
, u'\u3163', u'\u1d25', u' ', u'\u3141', u'\u314f', u'\u3139', u
'\u1d25', u'\u3147', u'\u3154', u'\u1d25', u' ', u'\u3147', u'\u
3163', u'\u1d25', u'\u3139', u'\u3161', u'\u1d25', u'\u3131', u'
\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u3139', u'\u1d25',
u',', u' ', u'\n', u' ', u' ', u'"', u'\u3145', u'\u3157', u'\u1
d25', u'\u3148', u'\u3153', u'\u1d25', u'\u3131', u'\u314f', u'\
u1d25', u' ', u'\u3147', u'\u3163', u'\u1d25', u'\u3139', u'\u31
61', u'\u1d25', u'\u3131', u'\u3163', u'\u1d25', u'\u3139', u'\u
3161', u'\u3139', u'\u1d25', u',', u' ', u'\n', u' ', u' ', u'"'
, u'\u3145', u'\u3157', u'\u1d25', u'\u3148', u'\u3153', u'\u1d2
5', u'\u3131', u'\u314f', u'\u1d25', u' ', u'\u3147', u'\u3153',
 u'\u1d25', u'\u3149', u'\u3163', u'\u1d25', u' ', u'\u3145', u'
\u314f', u'\u1d25', u'\u3139', u'\u314f', u'\u3141', u'\u1d25',
u'\u3147', u'\u3161', u'\u3139', u'\u1d25', u' ', u'\u3137', u'\
u3161', u'\u3139', u'\u1d25', u'\u3147', u'\u3161', u'\u1d25', u
'\u3134', u'\u3163', u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u
1d25', u' ', u'\u3141', u'\u314f', u'\u3139', u'\u1d25', u'\u314
7', u'\u3161', u'\u3139', u'\u1d25', u' ', u'\u3137', u'\u3161',
 u'\u3137', u'\u1d25', u'\u3131', u'\u3157', u'\u1d25', u' ', u'
\u3147', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u1d25',
u'\u3131', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u3139'
, u'\u1d25', u',', u' ', u'\n', u' ', u' ', u'"', u'\u3145', u'\
u3157', u'\u1d25', u'\u3148', u'\u3153', u'\u1d25', u'\u3131', u
'\u314f', u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u1d25', u'\u
3139', u'\u3161', u'\u1d25', u'\u3131', u'\u3163', u'\u1d25', u'
\u3139', u'\u3161', u'\u3139', u'\u1d25', u',', u' ', u'\n', u'
', u' ', u'"', u'\u3145', u'\u3157', u'\u1d25', u'\u3148', u'\u3
153', u'\u1d25', u'\u3131', u'\u314f', u'\u1d25', u' ', u'\u3147
', u'\u3153', u'\u1d25', u'\u3149', u'\u3163', u'\u1d25', u' ',
u'\u3145', u'\u314f', u'\u1d25', u'\u3139', u'\u314f', u'\u3141'
, u'\u1d25', u'\u3147', u'\u3161', u'\u3139', u'\u1d25', u' ', u
'\u3137', u'\u3161', u'\u3139', u'\u1d25', u'\u3147', u'\u3153',
 u'\u1d25', u' ', u'\u3131', u'\u314f', u'\u1d25', u'\u3145', u'
\u3163', u'\u1d25', u'\u3131', u'\u314f', u'\u1d25', u' ', u'\u3
147', u'\u314f', u'\u1d25', u'\u3134', u'\u3163', u'\u1d25', u'\
u3139', u'\u314f', u'\u1d25', u' ', u'\u314e', u'\u314f', u'\u1d
25', u'\u3147', u'\u3157', u'\u1d25', u'\u3134', u'\u3163', u'\u
1d25', u' ', u'\u3147', u'\u3163', u'\u1d25', u' ', u'\u3147', u
'\u3163', u'\u3139', u'\u1d25', u'\u3147', u'\u3161', u'\u3134',
 u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u1d25', u' ', u'\u314
1', u'\u3157', u'\u3141', u'\u1d25', u'\u3147', u'\u3163', u'\u1
d25', u' ', u'\u3147', u'\u3153', u'\u1d25', u'\u3149', u'\u3163
', u'\u1d25', u' ', u'\u3131', u'\u314f', u'\u1d25', u'\u3147',
u'\u3161', u'\u3139', u'\u1d25', u'\u3147', u'\u3161', u'\u3139'
, u'\u1d25', u' ', u'\n', u'\u3142', u'\u3157', u'\u1d25', u'\u3
134', u'\u3150', u'\u1d25', u'\u3147', u'\u3153', u'\u1d25', u'
', u'\u3147', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u31
41', u'\u1d25', u'\u3147', u'\u3161', u'\u3139', u'\u1d25', u' '
, u'\u3147', u'\u3163', u'\u1d25', u'\u3139', u'\u315c', u'\u1d2
5', u'\u3131', u'\u3157', u'\u1d25', u' ', u'\u3147', u'\u3163',
 u'\u3146', u'\u1d25', u'\u3147', u'\u3153', u'\u3146', u'\u1d25
', u'\u3137', u'\u314f', u'\u1d25', u'.', u' ', u'\n', u' ', u'
```

```
', u'\u3147', u'\u3163', u'\u1d25', u'\u3147', u'\u3154', u'\u1d
25', u' ', u'\u3137', u'\u3150', u'\u1d25', u'\u3137', u'\u314f'
, u'\u3142', u'\u1d25', u'\u314e', u'\u314f', u'\u1d25', u'\u313
1', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u3139', u'\u1
d25', u',', u' ', u'\n', u' ', u' ', u'"', u'\u3145', u'\u3157',
 u'\u1d25', u'\u3147', u'\u3160', u'\u1d25', u'\u3131', u'\u314f
', u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u1d25', u'\u3139',
u'\u3161', u'\u1d25', u'\u3131', u'\u3163', u'\u1d25', u'\u3139'
, u'\u3161', u'\u3139', u'\u1d25', u',', u' ', u'\n', u' ', u' '
, u'"', u'\u3147', u'\u3163', u'\u1d25', u'\u3148', u'\u3154', u
'\u1d25', u' ', u'\u3145', u'\u314f', u'\u1d25', u'\u3139', u'\u
314f', u'\u3141', u'\u1d25', u'\u3147', u'\u3163', u'\u1d25', u'
 ', u'\u3147', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u1
d25', u'\u3131', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\
u3139', u'\u1d25', u',', u' ', u'\n', u' ', u' ', u'"', u'\u3145
', u'\u3157', u'\u1d25', u'\u3148', u'\u3153', u'\u1d25', u'\u31
31', u'\u314f', u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u1d25'
, u'\u3139', u'\u3161', u'\u1d25', u'\u3131', u'\u3163', u'\u1d2
5', u'\u3139', u'\u3161', u'\u3139', u'\u1d25', u',', u' ', u'\n
', u' ', u' ', u'"', u'\u3147', u'\u3163', u'\u1d25', u'\u3148',
 u'\u3154', u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u1d25', u'
\u3147', u'\u3154', u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u1
d25', u'\u3139', u'\u3161', u'\u1d25', u'\u3131', u'\u3163', u'\
u1d25', u'\u3139', u'\u3161', u'\u3139', u'\u1d25', u',', u' ',
u'\n', u' ', u' ', u'"', u'\u3145', u'\u3157', u'\u1d25', u'\u31
47', u'\u3160', u'\u1d25', u'\u3134', u'\u3161', u'\u3134', u'\u
1d25', u' ', u'\u3147', u'\u3163', u'\u1d25', u'\u3141', u'\u316
3', u'\u1d25', u' ', u'\u3145', u'\u314f', u'\u1d25', u'\u3139',
 u'\u314f', u'\u3141', u'\u1d25', u'\u3147', u'\u3161', u'\u3139
', u'\u1d25', u' ', u'\u3137', u'\u3161', u'\u3139', u'\u1d25',
u'\u3147', u'\u3161', u'\u1d25', u'\u3134', u'\u3163', u'\u1d25'
, u' ', u'\u3147', u'\u3163', u'\u1d25', u' ', u'\u3141', u'\u31
4f', u'\u3139', u'\u1d25', u'\u3147', u'\u3161', u'\u3139', u'\u
1d25', u' ', u'\u3137', u'\u3161', u'\u3137', u'\u1d25', u'\u313
1', u'\u3157', u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u1d25',
 u'\u3139', u'\u3161', u'\u1d25', u'\u3131', u'\u3163', u'\u1d25
', u'\u3139', u'\u3161', u'\u3139', u'\u1d25', u',', u' ', u'\n'
, u' ', u' ', u'"', u'\u3145', u'\u3157', u'\u1d25', u'\u3148',
u'\u3153', u'\u1d25', u'\u3131', u'\u314f', u'\u1d25', u' ', u'\
u3147', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u1d25', u
'\u3131', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u3139',
 u'\u1d25', u',', u' ', u'\n', u' ', u' ', u'"', u'\u3145', u'\u
3157', u'\u1d25', u'\u3148', u'\u3153', u'\u1d25', u'\u3131', u'
\u314f', u'\u1d25', u' ', u'\u3147', u'\u3153', u'\u1d25', u'\u3
149', u'\u3163', u'\u1d25', u' ', u'\u3145', u'\u314f', u'\u1d25
', u'\u3139', u'\u314f', u'\u3141', u'\u1d25', u'\u3147', u'\u31
61', u'\u3139', u'\u1d25', u' ', u'\u3137', u'\u3161', u'\u3139'
, u'\u1d25', u'\u3147', u'\u3153', u'\u1d25', u' ', u'\u3131', u
'\u314f', u'\u1d25', u'\u3145', u'\u3163', u'\u1d25', u'\u3131',
 u'\u314f', u'\u1d25', u' ', u'\u3147', u'\u314f', u'\u1d25', u'
\u3134', u'\u3163', u'\u1d25', u'\u3139', u'\u314f', u'\u1d25',
u' ', u'\u314e', u'\u314f', u'\u1d25', u'\u3147', u'\u3157', u'\
u1d25', u'\u3134', u'\u3163', u'\u1d25', u' ', u'\u3147', u'\u31
```

```
63', u'\u1d25', u' ', u'\u3147', u'\u314f', u'\u1d25', u'\u3134'
, u'\u3163', u'\u1d25', u' ', u'\u3147', u'\u3153', u'\u1d25', u
'\u3149', u'\u3163', u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u
1d25', u'\u3139', u'\u3161', u'\u3139', u'\u1d25', u' ', u'\u314
5', u'\u314f', u'\u1d25', u'\u3139', u'\u314f', u'\u3141', u'\u1
d25', u'\u3147', u'\u3161', u'\u3139', u'\u1d25', u' ', u'\u3145
', u'\u314f', u'\u1d25', u'\u3139', u'\u314f', u'\u3147', u'\u1d
25', u'\u314e', u'\u314f', u'\u1d25', u'\u3147', u'\u3155', u'\u
1d25', u' ', u'\u3147', u'\u3163', u'\u1d25', u' ', u'\u3147', u
'\u3163', u'\u3139', u'\u1d25', u'\u3147', u'\u3161', u'\u3139',
 u'\u1d25', u' ', u'\n', u'\u3142', u'\u3157', u'\u1d25', u'\u31
34', u'\u3150', u'\u1d25', u'\u3147', u'\u3153', u'\u1d25', u' '
, u'\u3147', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u314
1', u'\u1d25', u'\u3147', u'\u3161', u'\u3139', u'\u1d25', u' ',
 u'\u3147', u'\u3163', u'\u1d25', u'\u3139', u'\u315c', u'\u1d25
', u'\u3131', u'\u3157', u'\u1d25', u' ', u'\u3147', u'\u3163',
u'\u3146', u'\u1d25', u'\u3147', u'\u3153', u'\u3146', u'\u1d25'
, u'\u3137', u'\u314f', u'\u1d25', u'.', u' ', u'\n', u' ', u' '
, u'\u3147', u'\u3163', u'\u1d25', u'\u3147', u'\u3154', u'\u1d2
5', u' ', u'\u3137', u'\u3150', u'\u1d25', u'\u3137', u'\u314f',
 u'\u3142', u'\u1d25', u'\u314e', u'\u314f', u'\u1d25', u'\u3131
', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u3139', u'\u1d
25', u',', u' ', u'\n', u' ', u' ', u'"', u'\u3145', u'\u3157',
u'\u1d25', u'\u3147', u'\u3160', u'\u1d25', u'\u3131', u'\u314f'
, u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u1d25', u'\u3139', u
'\u3161', u'\u1d25', u'\u3131', u'\u3163', u'\u1d25', u'\u3139',
 u'\u3161', u'\u3139', u'\u1d25', u',', u' ', u'\n', u' ', u' ',
 u'"', u'\u3147', u'\u3163', u'\u1d25', u'\u3148', u'\u3154', u'
\u1d25', u' ', u'\u3145', u'\u314f', u'\u1d25', u'\u3139', u'\u3
14f', u'\u3141', u'\u1d25', u'\u3147', u'\u3163', u'\u1d25', u'
', u'\u3147', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u1d
25', u'\u3131', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u
3139', u'\u1d25', u',', u' ', u'\n', u' ', u' ', u'"', u'\u3145'
, u'\u3157', u'\u1d25', u'\u3148', u'\u3153', u'\u1d25', u'\u313
1', u'\u314f', u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u1d25',
 u'\u3139', u'\u3161', u'\u1d25', u'\u3131', u'\u3163', u'\u1d25
', u'\u3139', u'\u3161', u'\u3139', u'\u1d25', u',', u' ', u'\n'
, u' ', u' ', u'"', u'\u3145', u'\u3157', u'\u1d25', u'\u3148',
u'\u3153', u'\u1d25', u'\u3131', u'\u314f', u'\u1d25', u' ', u'\
u3147', u'\u3153', u'\u1d25', u'\u3149', u'\u3163', u'\u1d25', u
' ', u'\u3145', u'\u314f', u'\u1d25', u'\u3139', u'\u314f', u'\u
3141', u'\u1d25', u'\u3147', u'\u3161', u'\u3139', u'\u1d25', u'
 ', u'\u3137', u'\u3161', u'\u3139', u'\u1d25', u'\u3147', u'\u3
153', u'\u1d25', u' ', u'\u3131', u'\u314f', u'\u1d25', u'\u3145
', u'\u3163', u'\u1d25', u'\u3131', u'\u314f', u'\u1d25', u' ',
u'\u3147', u'\u314f', u'\u1d25', u'\u3134', u'\u3163', u'\u1d25'
, u'\u3139', u'\u314f', u'\u1d25', u' ', u'\u314e', u'\u314f', u
'\u1d25', u'\u3147', u'\u3157', u'\u1d25', u'\u3134', u'\u3163',
 u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u1d25', u' ', u'\u314
7', u'\u3163', u'\u3139', u'\u1d25', u'\u3147', u'\u3161', u'\u3
134', u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u1d25', u' ', u'
\u3141', u'\u3157', u'\u3141', u'\u1d25', u'\u3147', u'\u3163',
u'\u1d25', u' ', u'\u3147', u'\u3153', u'\u1d25', u'\u3149', u'\
```

```
u3163', u'\u1d25', u' ', u'\u3131', u'\u314f', u'\u1d25', u'\u31
47', u'\u3161', u'\u3139', u'\u1d25', u'\u3147', u'\u3161', u'\u
3139', u'\u1d25', u' ', u'\n', u'\u3142', u'\u3157', u'\u1d25',
u'\u3134', u'\u3150', u'\u1d25', u'\u3147', u'\u3153', u'\u1d25'
, u' ', u'\u3147', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u
'\u3141', u'\u1d25', u'\u3147', u'\u3161', u'\u3139', u'\u1d25',
 u' ', u'\u3147', u'\u3163', u'\u1d25', u'\u3139', u'\u315c', u'
\u1d25', u'\u3131', u'\u3157', u'\u1d25', u' ', u'\u3147', u'\u3
163', u'\u3146', u'\u1d25', u'\u3147', u'\u3153', u'\u3146', u'\
u1d25', u'\u3137', u'\u314f', u'\u1d25', u'.', u' ', u'\n', u' '
, u' ', u'\u3147', u'\u3163', u'\u1d25', u'\u3147', u'\u3154', u
'\u1d25', u' ', u'\u3137', u'\u3150', u'\u1d25', u'\u3137', u'\u
314f', u'\u3142', u'\u1d25', u'\u314e', u'\u314f', u'\u1d25', u'
\u3131', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u3139',
u'\u1d25', u',', u' ', u'\n', u' ', u' ', u'"', u'\u3145', u'\u3
157', u'\u1d25', u'\u3147', u'\u3160', u'\u1d25', u'\u3131', u'\
u314f', u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u1d25', u'\u31
39', u'\u3161', u'\u3141', u'\u1d25', u'\u3147', u'\u3161', u'\u
3139', u'\u1d25', u' ', u'\u3137', u'\u3161', u'\u3139', u'\u1d2
5', u'\u3147', u'\u3153', u'\u1d25', u' ', u'\u3131', u'\u314f',
 u'\u1d25', u'\u3145', u'\u3163', u'\u1d25', u'\u3131', u'\u314f
', u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u3146', u'\u1d25',
u'\u3147', u'\u3161', u'\u1d25', u'\u3134', u'\u3163', u'\u1d25'
, u' ', u'\u3147', u'\u3163', u'\u1d25', u' ', u'\u3141', u'\u31
57', u'\u3141', u'\u1d25', u'\u3147', u'\u3163', u'\u1d25', u' '
, u'\u3147', u'\u3153', u'\u1d25', u'\u3149', u'\u3163', u'\u1d2
5', u' ', u'\u3131', u'\u314f', u'\u1d25', u'\u314e', u'\u3163',
 u'\u1d25', u' ', u'\u3147', u'\u3155', u'\u1d25', u'\u3131', u'
\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u3139', u'\u1d25',
u' ', u'\u3137', u'\u3161', u'\u3139', u'\u1d25', u'\u3147', u'\
u3161', u'\u3134', u'\u1d25', u' ', u'\u3131', u'\u3153', u'\u31
45', u'\u1d25', u'\u3147', u'\u3163', u'\u1d25', u' ', u'\u3147'
, u'\u3153', u'\u3144', u'\u1d25', u'\u3145', u'\u314f', u'\u1d2
5', u'\u3147', u'\u3157', u'\u1d25', u'\u3134', u'\u3163', u'\u1
d25', u' ', u'\u3147', u'\u3153', u'\u1d25', u'\u3149', u'\u3163
', u'\u1d25', u' ', u'\u3131', u'\u3161', u'\u1d25', u' ', u'\u3
147', u'\u3155', u'\u1d25', u'\u3131', u'\u3158', u'\u3134', u'\
u1d25', u'\u3147', u'\u3163', u'\u1d25', u' ', u'\u3147', u'\u31
53', u'\u3144', u'\u1d25', u'\u3134', u'\u3161', u'\u3134', u'\u
1d25', u' ', u'\u3131', u'\u3153', u'\u3145', u'\u1d25', u'\u314
7', u'\u3163', u'\u1d25', u'\u3139', u'\u314f', u'\u1d25', u' ',
 u'\u314e', u'\u314f', u'\u1d25', u'\u3134', u'\u314f', u'\u1d25
', u' ', u'\u3147', u'\u3163', u'\u1d25', u' ', u'\u3147', u'\u3
163', u'\u1d25', u'\u3139', u'\u3161', u'\u3141', u'\u1d25', u'\
u3147', u'\u3161', u'\u3139', u'\u1d25', u' ', u'\n', u'\u3147',
 u'\u3163', u'\u1d25', u'\u3139', u'\u315c', u'\u1d25', u'\u3147
', u'\u3153', u'\u3146', u'\u1d25', u'\u3137', u'\u314f', u'\u1d
25', u'.', u' ', u'\n', u' ', u' ', u'"', u'\u3147', u'\u3163',
u'\u1d25', u' ', u'\u3141', u'\u314f', u'\u3139', u'\u1d25', u'\
u3147', u'\u3161', u'\u3139', u'\u1d25', u' ', u'\u3137', u'\u31
61', u'\u3139', u'\u1d25', u'\u3147', u'\u3153', u'\u1d25', u' '
, u'\u3147', u'\u3157', u'\u1d25', u'\u3139', u'\u3161', u'\u1d2
5', u'\u3134', u'\u3163', u'\u1d25', u' ', u'\u3147', u'\u3163',
```

```
 u'\u1d25', u' ', u'\u3141', u'\u3157', u'\u3141', u'\u1d25', u'
\u3147', u'\u3163', u'\u1d25', u' ', u'\u3147', u'\u3153', u'\u1
d25', u'\u3149', u'\u3163', u'\u1d25', u' ', u'\u3147', u'\u3163
', u'\u1d25', u'\u3139', u'\u3161', u'\u1d25', u'\u3131', u'\u31
63', u'\u1d25', u'\u3139', u'\u3161', u'\u3139', u'\u1d25', u','
, u' ', u'\n', u' ', u' ', u'"', u'\u3145', u'\u3157', u'\u1d25'
, u'\u3148', u'\u3153', u'\u1d25', u'\u3131', u'\u314f', u'\u1d2
5', u' ', u'\u3147', u'\u3163', u'\u1d25', u'\u3139', u'\u3161',
 u'\u1d25', u'\u3131', u'\u3163', u'\u1d25', u'\u3139', u'\u3161
', u'\u3139', u'\u1d25', u',', u' ', u'\n', u' ', u' ', u'"', u'
\u3145', u'\u3157', u'\u1d25', u'\u3148', u'\u3153', u'\u1d25',
u'\u3131', u'\u314f', u'\u1d25', u' ', u'\u3147', u'\u3153', u'\
u1d25', u'\u3149', u'\u3163', u'\u1d25', u' ', u'\u3145', u'\u31
4f', u'\u1d25', u'\u3139', u'\u314f', u'\u3141', u'\u1d25', u'\u
3147', u'\u3161', u'\u3139', u'\u1d25', u' ', u'\u3137', u'\u316
1', u'\u3139', u'\u1d25', u'\u3147', u'\u3161', u'\u1d25', u'\u3
134', u'\u3163', u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u1d25
', u' ', u'\u3141', u'\u314f', u'\u3139', u'\u1d25', u'\u3147',
u'\u3161', u'\u3139', u'\u1d25', u' ', u'\u3137', u'\u3161', u'\
u3137', u'\u1d25', u'\u3131', u'\u3157', u'\u1d25', u' ', u'\u31
47', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u1d25', u'\u
3131', u'\u3163', u'\u1d25', u'\u3139', u'\u3161', u'\u3139', u'
\u1d25', u',', u' ', u'\n', u' ', u' ', u'"', u'\u3145', u'\u315
7', u'\u1d25', u'\u3148', u'\u3153', u'\u1d25', u'\u3131', u'\u3
14f', u'\u1d25', u' ', u'\u3147', u'\u3163', u'\u1d25', u'\u3139
', u'\u3161', u'\u1d25', u'\u3131', u'\u3163', u'\u1d25', u'\u31
39', u'\u3161', u'\u3139', u'\u1d25', u',', u' ', u'\n', u' ', u
' ', u'"', u'\u3145', u'\u3157', u'\u1d25', u'\u3148', u'\u3153'
, u'\u1d25', u'\u3131', u'\u314f', u'\u1d25', u' ', u'\u3147', u
'\u3153', u'\u1d25', u'\u3149', u'\u3163', u'\u1d25', u' ', u'\u
3145', u'\u314f', u'\u1d25', u'\u3139', u'\u314f', u'\u3141', u'
\u1d25', u'\u3147', u'\u3161', u'\u3139', u'\u1d25', u' ', u'\u3
137', u'\u3161', u'\u3139', u'\u1d25', u'\u3147', u'\u3153', u'\
u1d25', u' ', u'\u3131', u'\u314f', u'\u1d25', u'\u3145', u'\u31
63', u'\u1d25', u'\u3131']

-- RESULT --
누구 소리에 서 있을 것이로다."
  이어서 서울에 들어가니 어찌 이 말을 듣고 있으니 이 말을 듣고 있으니 이 말을
 듣고 있었다.
  이에 대답하기를,
  "소유가 다시 이르기를,
  "이제 이에 이르기를,
  "소저가 어찌 이르기를,
  "소저는 이 말을 듣고 이르기를,
  "소저가 어찌 사람을 들어 가서 그 아름다운 일이었다.
  이 말에 이르기를,
  "소저가 이르기를,
  "소저가 어찌 사람을 들으니 이 말을 듣고 이르기를,
  "소저가 이르기를,
  "소저가 어찌 사람을 들어 가시가 아니라 하오니 이 일은 이 몸이 어찌 가을을
보내어 이름을 이루고 있었다.
  이에 대답하기를,
```

```
  "소유가 이르기를,
  "이제 사람이 이르기를,
  "소저가 이르기를,
  "이제 이에 이르기를,
  "소유는 이미 사람을 들으니 이 말을 듣고 이르기를,
  "소저가 이르기를,
  "소저가 어찌 사람을 들어 가시가 아니라 하오니 이 아니 어찌 이를 사람을 사랑
하여 이 일을
보내어 이름을 이루고 있었다.
  이에 대답하기를,
  "소유가 이르기를,
  "이제 사람이 이르기를,
  "소저가 이르기를,
  "소저가 어찌 사람을 들어 가시가 아니라 하오니 이 일은 이 몸이 어찌 가을을
보내어 이름을 이루고 있었다.
  이에 대답하기를,
  "소유가 이름을 들어 가시가 있으니 이 몸이 어찌 가히 여기를 들은 것이 없사오
니 어찌 그 여관이 없는 것이라 하나 이 이름을
이루었다.
  "이 말을 들어 오르니 이 몸이 어찌 이르기를,
  "소저가 이르기를,
  "소저가 어찌 사람을 들으니 이 말을 듣고 이르기를,
  "소저가 이르기를,
  "소저가 어찌 사람을 들어 가시
```

Word Embedding (Word2Vec)

Very simple word2vec example @ nlintz's tutoral

```
import collections
import numpy as np
import tensorflow as tf
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

print ("Packages loaded.")
```

```
Packages loaded.
```

Configuration

```
# Configuration
batch_size     = 20
embedding_size = 2      # This is just for visualization
num_sampled    = 15     # Number of negative examples to sample.
```

Sentences, we will use

```
# Sample sentences
sentences = ["the quick brown fox jumped over the lazy dog",
            "I love cats and dogs",
            "we all love cats and dogs",
            "cats and dogs are great",
            "sung likes cats",
            "she loves dogs",
            "cats can be very independent",
            "cats are great companions when they want to be",
            "cats are playful",
            "cats are natural hunters",
            "It's raining cats and dogs",
            "dogs and cats love sung"]
# 'sentences' is 'list'
print ("'sentences' is %s and length is %d."
       % (type(sentences), len(sentences)))
```

```
'sentences' is <type 'list'> and length is 12.
```

sentences to words and count

words: list of all words (just concatenation)

```
words = " ".join(sentences).split()
print ("'words' is %s and length is %d." % (type(words), len(wor
ds)))
print (words)
```

```
'words' is <type 'list'> and length is 62.
['the', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy'
, 'dog', 'I', 'love', 'cats', 'and', 'dogs', 'we', 'all', 'love'
, 'cats', 'and', 'dogs', 'cats', 'and', 'dogs', 'are', 'great',
'sung', 'likes', 'cats', 'she', 'loves', 'dogs', 'cats', 'can',
'be', 'very', 'independent', 'cats', 'are', 'great', 'companions
', 'when', 'they', 'want', 'to', 'be', 'cats', 'are', 'playful',
 'cats', 'are', 'natural', 'hunters', "It's", 'raining', 'cats',
 'and', 'dogs', 'dogs', 'and', 'cats', 'love', 'sung']
```

count: list of pairs, each pair consists of 'cats', 10

```
count = collections.Counter(words).most_common()
print ("'count' is %s and length is %d." % (type(count), len(cou
nt)))
print (("Word count of top five is %s") % (count[:5]))
print (count)
```

```
'count' is <type 'list'> and length is 35.
Word count of top five is [('cats', 10), ('dogs', 6), ('and', 5)
, ('are', 4), ('love', 3)]
[('cats', 10), ('dogs', 6), ('and', 5), ('are', 4), ('love', 3),
 ('be', 2), ('sung', 2), ('great', 2), ('the', 2), ('raining', 1
), ('all', 1), ('when', 1), ('over', 1), ('we', 1), ('playful',
1), ('want', 1), ('to', 1), ('jumped', 1), ('hunters', 1), ('com
panions', 1), ('fox', 1), ('very', 1), ("It's", 1), ('can', 1),
('brown', 1), ('lazy', 1), ('I', 1), ('independent', 1), ('they'
, 1), ('natural', 1), ('dog', 1), ('she', 1), ('loves', 1), ('qu
ick', 1), ('likes', 1)]
```

See what's in the 'words' and 'count'

```
print (words[0:5])
print (count[0:3])
```

```
['the', 'quick', 'brown', 'fox', 'jumped']
[('cats', 10), ('dogs', 6), ('and', 5)]
```

Build dictionaries

```
rdic = [i[0] for i in count] #reverse dic, idx -> word
dic = {w: i for i, w in enumerate(rdic)} #dic, word -> id
voc_size = len(dic) # Number of vocabulary
print ("'rdic' is %s and length is %d." % (type(rdic), len(rdic)
))
print ("'dic' is %s and length is %d." % (type(dic), len(dic)))
```

```
'rdic' is <type 'list'> and length is 35.
'dic' is <type 'dict'> and length is 35.
```

```
print (rdic)
```

```
['cats', 'dogs', 'and', 'are', 'love', 'be', 'sung', 'great', 't
he', 'raining', 'all', 'when', 'over', 'we', 'playful', 'want',
'to', 'jumped', 'hunters', 'companions', 'fox', 'very', "It's",
'can', 'brown', 'lazy', 'I', 'independent', 'they', 'natural', '
dog', 'she', 'loves', 'quick', 'likes']
```

```
print (dic)
```

```
{'and': 2, 'raining': 9, 'all': 10, 'love': 4, 'brown': 24, 'whe
n': 11, 'over': 12, 'lazy': 25, 'playful': 14, 'are': 3, 'want':
 15, 'sung': 6, 'jumped': 17, 'hunters': 18, 'companions': 19, '
fox': 20, 'to': 16, 'cats': 0, "It's": 22, 'dogs': 1, 'she': 31,
 'be': 5, 'we': 13, 'very': 21, 'independent': 27, 'they': 28, '
natural': 29, 'great': 7, 'I': 26, 'dog': 30, 'can': 23, 'loves'
: 32, 'quick': 33, 'the': 8, 'likes': 34}
```

See what's in the 'rdic' and 'rdic'

```
print (rdic[0])
print (dic['cats'])
```

```
cats
0
```

Make indexed word data (ordered)

```
data = [dic[word] for word in words]
print ("'data' is %s and length is %d." % (type(data), len(data)
))
print('Sample data: numbers: %s / words: %s'% (data[:10], [rdic[
t] for t in data[:10]]))
```

```
'data' is <type 'list'> and length is 62.
Sample data: numbers: [8, 33, 24, 20, 17, 12, 8, 25, 30, 26] / w
ords: ['the', 'quick', 'brown', 'fox', 'jumped', 'over', 'the',
'lazy', 'dog', 'I']
```

```
# See what's in the data
print (data)
```

```
[8, 33, 24, 20, 17, 12, 8, 25, 30, 26, 4, 0, 2, 1, 13, 10, 4, 0,
 2, 1, 0, 2, 1, 3, 7, 6, 34, 0, 31, 32, 1, 0, 23, 5, 21, 27, 0,
3, 7, 19, 11, 28, 15, 16, 5, 0, 3, 14, 0, 3, 29, 18, 22, 9, 0, 2
, 1, 1, 2, 0, 4, 6]
```

Let's make a training data for window size 1 for simplicity

```
# ([the, brown], quick), ([quick, fox], brown), ([brown, jumped]
, fox),
cbow_pairs = [];
for i in range(1, len(data)-1) :
    cbow_pairs.append([[data[i-1], data[i+1]], data[i]]);
print('Context pairs: %s' % (cbow_pairs[:10]))
```

```
Context pairs: [[[8, 24], 33], [[33, 20], 24], [[24, 17], 20], [
[20, 12], 17], [[17, 8], 12], [[12, 25], 8], [[8, 30], 25], [[25
, 26], 30], [[30, 4], 26], [[26, 0], 4]]
```

See type and length of 'cbow_pairs'

```
print ("'cbow_pairs' is %s and length is %d."
       % (type(cbow_pairs), len(cbow_pairs)))
```

```
'cbow_pairs' is <type 'list'> and length is 60.
```

Let's make skip-gram pairs

```
# (quick, the), (quick, brown), (brown, quick), (brown, fox), ...

skip_gram_pairs = [];
for c in cbow_pairs:
    skip_gram_pairs.append([c[1], c[0][0]])
    skip_gram_pairs.append([c[1], c[0][1]])

print ("'skip_gram_pairs' is %s and length is %d."
       % (type(skip_gram_pairs), len(skip_gram_pairs)))
print('skip-gram pairs', skip_gram_pairs[:5])
```

```
'skip_gram_pairs' is <type 'list'> and length is 120.
('skip-gram pairs', [[33, 8], [33, 24], [24, 33], [24, 20], [20,
 24]])
```

```
def generate_batch(size):
    assert size < len(skip_gram_pairs)
    x_data=[]
    y_data = []
    r = np.random.choice(range(len(skip_gram_pairs)), size, repl
ace=False)
    for i in r:
        x_data.append(skip_gram_pairs[i][0])  # n dim
        y_data.append([skip_gram_pairs[i][1]])  # n, 1 dim
    return x_data, y_data

# generate_batch test
print ('Batches (x, y)', generate_batch(3))
```

```
('Batches (x, y)', ([3, 8, 7], [[7], [25], [3]]))
```

Construct network

```
# Input data
train_inputs = tf.placeholder(tf.int32, shape=[batch_size])
# need to shape [batch_size, 1] for nn.nce_loss
train_labels = tf.placeholder(tf.int32, shape=[batch_size, 1])
# missing GPU implementation?
with tf.device('/cpu:0'):
    # Look up embeddings for inputs.
    embeddings = tf.Variable(
        tf.random_uniform([voc_size, embedding_size], -1.0, 1.0)
)
    embed = tf.nn.embedding_lookup(embeddings, train_inputs) # l
ookup table

# Construct the variables for the NCE loss
nce_weights = tf.Variable(
    tf.random_uniform([voc_size, embedding_size], -1.0, 1.0))
nce_biases = tf.Variable(tf.zeros([voc_size]))

# Compute the average NCE loss for the batch.
loss = tf.reduce_mean(
    tf.nn.nce_loss(nce_weights, nce_biases, embed, train_labels,
                   num_sampled, voc_size))

# Use the adam optimizer
train_op = tf.train.AdamOptimizer(0.01).minimize(loss)
print ("Network ready")
```

```
Network ready
```

Run!

```
# Launch the graph in a session
with tf.Session() as sess:
    # Initializing all variables
    tf.initialize_all_variables().run()

    for step in range(3000):
        batch_inputs, batch_labels = generate_batch(batch_size)
        _, loss_val = sess.run([train_op, loss],
                feed_dict={train_inputs: batch_inputs, train_lab
els: batch_labels})
        if step % 500 == 0:
            print("Loss at %d: %.5f" % (step, loss_val))
            # Report the loss

    # Final embeddings are ready for you to use.
    # Need to normalize for practical use
    trained_embeddings = embeddings.eval()
```

```
Loss at 0: 18.82132
Loss at 500: 3.69854
Loss at 1000: 3.22232
Loss at 1500: 2.94147
Loss at 2000: 2.90176
Loss at 2500: 2.71721
```

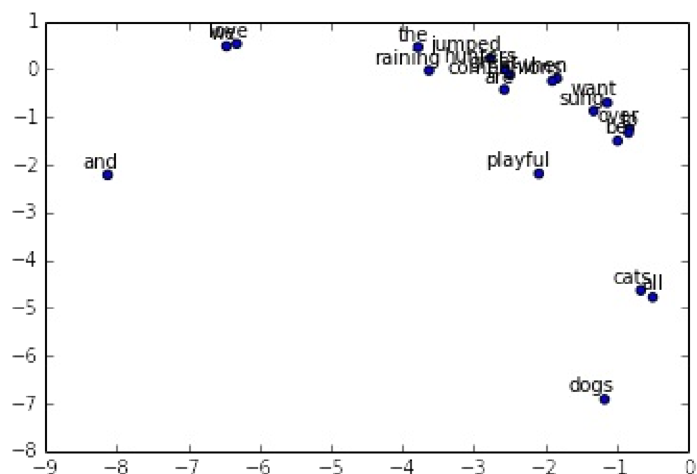
```
trained_embeddings.shape
```

```
(35, 2)
```

Plot results

```
# Show word2vec if dim is 2
if trained_embeddings.shape[1] == 2:
    labels = rdic[:20] # Show top 20 words
    for i, label in enumerate(labels):
        x, y = trained_embeddings[i,:]
        plt.scatter(x, y)
        plt.annotate(label, xy=(x, y), xytext=(5, 2),
            textcoords='offset points', ha='right', va='bottom')
    plt.show()
```

```
/usr/lib/pymodules/python2.7/matplotlib/collections.py:548: Futu
reWarning: elementwise comparison failed; returning scalar inste
ad, but in the future will perform elementwise comparison
  if self._edgecolors == 'face':
```



Word2vec basic

```
import collections
import math
import os
import random
import zipfile
import numpy as np
from six.moves import urllib
from six.moves import xrange  # pylint: disable=redefined-builtin

from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import tensorflow as tf
%matplotlib inline
print ("Packages loaded")
```

```
Packages loaded
```

1. Download the text and make corpus (set of words)

Download (or reuse) the text file that we will use

```
folder_dir  = "data"
file_name   = "text8.zip"
file_path   = os.path.join(folder_dir, file_name)
url         = 'http://mattmahoney.net/dc/'
if not os.path.exists(file_path):
    print ("No file found. Start downloading")
    downfilename, _ = urllib.request.urlretrieve(
        url + file_name, file_path)
    print ("'%s' downloaded" % (downfilename))
else:
    print ("File already exists")
```

```
File already exists
```

Check we have correct data

```
statinfo = os.stat(file_path)
expected_bytes = 31344016
if statinfo.st_size == expected_bytes:
    print ("I guess we have correct file at '%s'" % (file_path))
else:
    print ("Something's wrong with the file at '%s'" % (file_pat
h))
```

```
I guess we have correct file at 'data/text8.zip'
```

Unzip the file

```
def read_data(filename):
    with zipfile.ZipFile(filename) as f:
        data = f.read(f.namelist()[0]).split()
    return data
```

```
words = read_data(file_path)
print ("Type of 'words' is %s / Length is %d "
       % (type(words), len(words)))
print ("'words' look like \n %s" %(words[0:100]))
```

```
Type of 'words' is <type 'list'> / Length is 17005207
'words' look like
 ['anarchism', 'originated', 'as', 'a', 'term', 'of', 'abuse', '
first', 'used', 'against', 'early', 'working', 'class', 'radical
s', 'including', 'the', 'diggers', 'of', 'the', 'english', 'revo
lution', 'and', 'the', 'sans', 'culottes', 'of', 'the', 'french'
, 'revolution', 'whilst', 'the', 'term', 'is', 'still', 'used',
'in', 'a', 'pejorative', 'way', 'to', 'describe', 'any', 'act',
'that', 'used', 'violent', 'means', 'to', 'destroy', 'the', 'org
anization', 'of', 'society', 'it', 'has', 'also', 'been', 'taken
', 'up', 'as', 'a', 'positive', 'label', 'by', 'self', 'defined'
, 'anarchists', 'the', 'word', 'anarchism', 'is', 'derived', 'fr
om', 'the', 'greek', 'without', 'archons', 'ruler', 'chief', 'ki
ng', 'anarchism', 'as', 'a', 'political', 'philosophy', 'is', 't
he', 'belief', 'that', 'rulers', 'are', 'unnecessary', 'and', 's
hould', 'be', 'abolished', 'although', 'there', 'are', 'differin
g']
```

2. Make a dictionary with fixed length (using UNK token)

Count the words

```
vocabulary_size = 50000
count = [['UNK', -1]]
count.extend(collections.Counter(words)
             .most_common(vocabulary_size - 1)) # -1 is for UNK
print ("Type of 'count' is %s / Length is %d " % (type(count), l
en(count)))
print ("'count' looks like \n %s" % (count[0:10]))
```

```
Type of 'count' is <type 'list'> / Length is 50000
'count' looks like
 [['UNK', -1], ('the', 1061396), ('of', 593677), ('and', 416629)
, ('one', 411764), ('in', 372201), ('a', 325873), ('to', 316376)
, ('zero', 264975), ('nine', 250430)]
```

Make a dictionary

```
dictionary = dict()
for word, _ in count:
    dictionary[word] = len(dictionary)
print ("Type of 'dictionary' is %s / Length is %d "
       % (type(dictionary), len(dictionary)))
```

```
Type of 'dictionary' is <type 'dict'> / Length is 50000
```

Make a reverse dictionary

```
reverse_dictionary = dict(zip(dictionary.values(), dictionary.ke
ys()))
print ("Type of 'reverse_dictionary' is %s / Length is %d "
       % (type(reverse_dictionary), len(reverse_dictionary)))
```

```
Type of 'reverse_dictionary' is <type 'dict'> / Length is 50000
```

Make data

```
data = list()
unk_count = 0
for word in words:
    if word in dictionary:
        index = dictionary[word]
    else:
        index = 0  # dictionary['UNK']
        unk_count += 1
    data.append(index)
count[0][1] = unk_count
# del words  # Hint to reduce memory.
```

'dictionary' converts word to index

'reverse_dictionary' converts index to word

```
print ("Most common words (+UNK) are: %s" % (count[:5]))
```

```
Most common words (+UNK) are: [['UNK', 418391], ('the', 1061396)
, ('of', 593677), ('and', 416629), ('one', 411764)]
```

Data (in indices)

```
print ("Sample data: %s" % (data[:10]))
```

```
Sample data: [5239, 3084, 12, 6, 195, 2, 3137, 46, 59, 156]
```

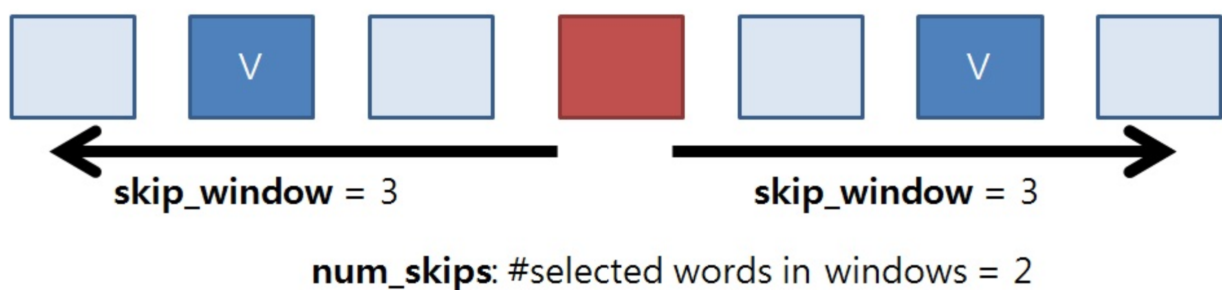
Convert to char (which we can read)

```
print ("Sample data corresponds to\n__________________")
for i in range(10):
    print ("%d->%s" % (data[i], reverse_dictionary[data[i]]))
```

```
Sample data corresponds to
__________________
5239->anarchism
3084->originated
12->as
6->a
195->term
2->of
3137->abuse
46->first
59->used
156->against
```

Batch-generating function for skip-gram model

- Skip-gram (one word to one word) => Can generate more training data



```
data_index = 0
def generate_batch(batch_size, num_skips, skip_window):
    global data_index
    assert batch_size % num_skips == 0
    assert num_skips <= 2 * skip_window
    batch  = np.ndarray(shape=(batch_size),    dtype=np.int32)
    labels = np.ndarray(shape=(batch_size, 1), dtype=np.int32)
    span = 2 * skip_window + 1 # [ skip_window target skip_windo
w ]
    buffer = collections.deque(maxlen=span)
    for _ in range(span):
        buffer.append(data[data_index])
        data_index = (data_index + 1) % len(data)
    for i in range(batch_size // num_skips): # '//' makes the re
sult an integer, e.g., 7//3 = 2
        target = skip_window
        targets_to_avoid = [ skip_window ]
        for j in range(num_skips):
            while target in targets_to_avoid:
                target = random.randint(0, span - 1)
            targets_to_avoid.append(target)
            batch[i * num_skips + j] = buffer[skip_window]
            labels[i * num_skips + j, 0] = buffer[target]
        buffer.append(data[data_index])
        data_index = (data_index + 1) % len(data)
    return batch, labels
```

Examples for generating batch and labels

```
data_index = 0
batch, labels = generate_batch(batch_size=8, num_skips=2, skip_w
indow=1)
print ("Type of 'batch' is %s / Length is %d "
       % (type(batch), len(batch)))
print ("Type of 'labels' is %s / Length is %d "
       % (type(labels), len(labels)))
```

```
Type of 'batch' is <type 'numpy.ndarray'> / Length is 8
Type of 'labels' is <type 'numpy.ndarray'> / Length is 8
```

```
print ("'batch' looks like \n %s" % (batch))
```

```
'batch' looks like
 [3084 3084   12   12    6    6  195  195]
```

```
print ("'labels' looks like \n %s" % (labels))
```

```
'labels' looks like
 [[5239]
 [  12]
 [3084]
 [   6]
 [ 195]
 [  12]
 [   2]
 [   6]]
```

```
for i in range(8):
    print ("%d -> %d"
           % (batch[i], labels[i, 0])),
    print ("\t%s -> %s"
           % (reverse_dictionary[batch[i]]
              , reverse_dictionary[labels[i, 0]]))
```

```
3084 -> 5239     originated -> anarchism
3084 -> 12     originated -> as
12 -> 3084     as -> originated
12 -> 6     as -> a
6 -> 195     a -> term
6 -> 12     a -> as
195 -> 2     term -> of
195 -> 6     term -> a
```

3. Build a Skip-Gram Model

```
batch_size     = 128
embedding_size = 128       # Dimension of the embedding vector.
skip_window    = 1         # How many words to consider left and
 right.
num_skips      = 2         # How many times to reuse an input
print ("Parameters ready")
```

```
Parameters ready
```

```
# Random validation set to sample nearest neighbors.
valid_size     = 32         # Random set of words to evaluate sim
ilarity
valid_window   = 200        # Only pick validation samples in the
 top 200
valid_examples = np.random.choice(valid_window, valid_size, repl
ace=False)

print (valid_examples)
```

```
[ 31  89  64  57 122 178 173  13  50 176  12 188   0  19 168  44
  59 130
 126  75 187 161  10 194 119 131   5  69 150 165   6  70]
```

Define network

```
# Construct the word2vec model
train_inputs   = tf.placeholder(tf.int32, shape=[batch_size])
train_labels   = tf.placeholder(tf.int32, shape=[batch_size, 1])
valid_dataset  = tf.constant(valid_examples, dtype=tf.int32)

# Look up embeddings for inputs. (vocabulary_size = 50,000)
with tf.variable_scope("EMBEDDING"):
    with tf.device('/cpu:0'):
        embeddings = tf.Variable(
            tf.random_uniform([vocabulary_size, embedding_size]
                              , -1.0, 1.0))
        embed = tf.nn.embedding_lookup(embeddings, train_inputs)

# Construct the variables for the NCE loss
with tf.variable_scope("NCE_WEIGHT"):
    nce_weights = tf.Variable(
        tf.truncated_normal([vocabulary_size, embedding_size],
                            stddev=1.0 / math.sqrt(embedding_siz
e)))
    nce_biases = tf.Variable(tf.zeros([vocabulary_size]))
print ("Network ready")
```

```
Network ready
```

Define functions

```
with tf.device('/cpu:0'):
    # Loss function
    num_sampled = 64        # Number of negative examples to sam
ple.
    loss = tf.reduce_mean(
        tf.nn.nce_loss(nce_weights, nce_biases, embed
                       , train_labels, num_sampled, vocabulary_s
ize))
    # Optimizer
    optm = tf.train.GradientDescentOptimizer(1.0).minimize(loss)
    # Similarity measure (important)
    norm = tf.sqrt(tf.reduce_sum(tf.square(embeddings), 1, keep_
dims=True))
    normalized_embeddings = embeddings / norm
    valid_embeddings = tf.nn.embedding_lookup(normalized_embeddi
ngs
                        , valid_dataset)
    siml = tf.matmul(valid_embeddings, normalized_embeddings
                        , transpose_b=True)

print ("Functions Ready")
```

```
Functions Ready
```

4. Train a Skip-Gram Model

```
# Train!
sess = tf.Session()
sess.run(tf.initialize_all_variables())
summary_writer = tf.train.SummaryWriter('/tmp/tf_logs/word2vec',
 graph=sess.graph)
average_loss = 0

num_steps = 100001
for iter in xrange(num_steps):
    batch_inputs, batch_labels = generate_batch(batch_size, num_
skips, skip_window)
    feed_dict = {train_inputs : batch_inputs, train_labels : bat
ch_labels}
    _, loss_val = sess.run([optm, loss], feed_dict=feed_dict)
    average_loss += loss_val

    if iter % 2000 == 0:
        average_loss /= 2000
        print ("Average loss at step %d is %.3f" % (iter, averag
e_loss))

    if iter % 10000 == 0:
        siml_val = sess.run(siml)
        for i in xrange(valid_size): # Among valid set
            valid_word = reverse_dictionary[valid_examples[i]]
            top_k = 6 # number of nearest neighbors
            nearest = (-siml_val[i, :]).argsort()[1:top_k+1]
            log_str = "Nearest to '%s':" % valid_word
            for k in xrange(top_k):
                close_word = reverse_dictionary[nearest[k]]
                log_str = "%s '%s'," % (log_str, close_word)
            print(log_str)

# Final embeding
final_embeddings = sess.run(normalized_embeddings)
```

```
Average loss at step 0 is 0.141
Nearest to 'an': 'yorktown', 'fux', 'katydids', 'leary', 'interr
uption', 'detox',
Nearest to 'called': 'wong', 'germaine', 'electrophilic', 'kilpa
trick', 'alkenes', 'retained',
Nearest to 'american': 'mq', 'reef', 'guang', 'seas', 'puma', 'r
q',
Nearest to 'who': 'scarred', 'directory', 'spate', 'denison', 'i
ncensed', 'adipose',
Nearest to 'c': 'chapman', 'charlize', 'lading', 'ethnographers'
, 'stingray', 'vainly',
Nearest to 'set': 'emptive', 'menelik', 'headed', 'unsubstantiat
ed', 'churchyard', 'horticultural',
Nearest to 'different': 'unreleased', 'arameans', 'psychologist'
```

```
, 'earmarked', 'phylum', 'ayer',
Nearest to 'eight': 'specificity', 'leftover', 'fulfil', 'urgell
', 'oak', 'columba',
Nearest to 'all': 'transport', 'linspire', 'eo', 'donatello', 'g
eorgi', 'picturesque',
Nearest to 'do': 'megabits', 'condorcet', 'landau', 'coincidenta
l', 'ransom', 'proper',
Nearest to 'as': 'yesterday', 'hw', 'solace', 'modeled', 'hats',
 'conant',
Nearest to 'law': 'gleaned', 'fickle', 'expressive', 'gory', 'di
em', 'threatens',
Nearest to 'UNK': 'probate', 'guests', 'determines', 'dave', 'ub
iquity', 'inks',
Nearest to 'by': 'parton', 'mccann', 'mahoney', 'diasporas', 'be
nefiting', 'nieuwe',
Nearest to 'large': 'towers', 'multiculturalism', 'endowment', '
enhancing', 'brainwashing', 'eia',
Nearest to 'their': 'ecc', 'somber', 'quaternary', 'schopenhauer
', 'trombone', 'brethren',
Nearest to 'used': 'shrunk', 'fixtures', 'ultraviolet', 'volley'
, 'hab', 'spiced',
Nearest to 'british': 'equinox', 'shandy', 'socialism', 'revert'
, 'zeeland', 'progs',
Nearest to 'x': 'callous', 'envoys', 'gamecube', 'unintelligent'
, 'rosicrucian', 'participated',
Nearest to 'd': 'francesco', 'schumacher', 'creighton', 'tubas',
 'indulgence', 'dip',
Nearest to 'king': 'circumscribed', 'niacin', 'inter', 'bluescre
en', 'catapult', 'farnese',
Nearest to 'based': 'oj', 'odi', 'fubar', 'jogaila', 'enables',
'refrigerated',
Nearest to 'two': 'orbits', 'hostel', 'skeleton', 'factories', '
borghese', 'panned',
Nearest to 'international': 'da', 'confidential', 'hiv', 'rhythm
ic', 'incipient', 'talkie',
Nearest to 'him': 'reelection', 'trivia', 'counters', 'lsch', 'v
isual', 'seaport',
Nearest to 'year': 'overwhelming', 'astaire', 'conclude', 'donne
r', 'honky', 'objects',
Nearest to 'in': 'events', 'golem', 'elbert', 'lung', 'rsc', 'co
ordinated',
Nearest to 'may': 'sur', 'formalized', 'inasmuch', 'telekinetic'
, 'bisexual', 'duchamp',
Nearest to 'g': 'aruban', 'africans', 'jason', 'perutz', 'stun',
 'hahn',
Nearest to 'de': 'compactification', 'brugge', 'disarming', 'plo
tted', 'env', 'fielder',
Nearest to 'a': 'ally', 'motorola', 'concrete', 'catcher', 'neck
s', 'mortimer',
Nearest to 'than': 'karts', 'gwynedd', 'hesse', 'agglutinative',
 'there', 'ensues',
Average loss at step 2000 is 114.071
Average loss at step 4000 is 52.629
```

```
Average loss at step 6000 is 33.005
Average loss at step 8000 is 23.590
Average loss at step 10000 is 17.952
Nearest to 'an': 'the', 'overseas', 'protein', 'interruption', '
victoriae', 'mya',
Nearest to 'called': 'gland', 'within', 'gb', 'much', 'retained'
, 'consists',
Nearest to 'american': 'olympics', 'methadone', 'linguistic', 't
echnologically', 'while', 'spider',
Nearest to 'who': 'and', 'directory', 'named', 'attacked', 'mode
', 'ruth',
Nearest to 'c': 'victoriae', 'accounts', 'mathbf', 'reginae', 'a
ustin', 'passion',
Nearest to 'set': 'reginae', 'shortcuts', 'animals', 'few', 'mem
bers', 'majority',
Nearest to 'different': 'mya', 'unreleased', 'cl', 'phylum', 'ps
ychologist', 'reginae',
Nearest to 'eight': 'nine', 'zero', 'six', 'reginae', 'agave', '
vs',
Nearest to 'all': 'transport', 'beginning', 'bestseller', 'cl',
'waugh', 'lindbergh',
Nearest to 'do': 'vs', 'proper', 'assistant', 'symbols', 'reflec
t', 'christian',
Nearest to 'as': 'in', 'for', 'and', 'by', 'irish', 'asterism',
Nearest to 'law': 'collapse', 'cc', 'soviet', 'accordion', 'badg
es', 'kyoto',
Nearest to 'UNK': 'and', 'alpina', 'mengele', 'the', 'reginae',
'one',
Nearest to 'by': 'in', 'and', 'as', 'at', 'gland', 'was',
Nearest to 'large': 'towers', 'gland', 'fricatives', 'conducted'
, 'greatest', 'charlie',
Nearest to 'their': 'reginae', 'schopenhauer', 'afghani', 'the',
 'sabotage', 'implicit',
Nearest to 'used': 'spiced', 'mystery', 'mathbf', 'sacred', 'min
ing', 'fixtures',
Nearest to 'british': 'anatomy', 'socialism', 'equinox', 'victor
iae', 'astor', 'five',
Nearest to 'x': 'phi', 'right', 'participated', 'modern', 'count
ries', 'video',
Nearest to 'd': 'finalist', 'avoid', 'tubing', 'vs', 'francesco'
, 'schumacher',
Nearest to 'king': 'inter', 'thompson', 'negligible', 'moves', '
alpina', 'wisconsin',
Nearest to 'based': 'reginae', 'chemist', 'dim', 'directory', 't
raditionally', 'agave',
Nearest to 'two': 'one', 'reginae', 'vs', 'austin', 'gollancz',
'three',
Nearest to 'international': 'da', 'hiv', 'confidential', 'access
', 'six', 'robots',
Nearest to 'him': 'infectious', 'trivia', 'spontaneous', 'seapor
t', 'directors', 'visual',
Nearest to 'year': 'conclude', 'objects', 'overwhelming', 'clear
', 'responsibility', 'vs',
```

```
Nearest to 'in': 'and', 'of', 'mya', 'with', 'on', 'by',
Nearest to 'may': 'sur', 'cannot', 'tubing', 'formalized', 'indi
vidualist', 'faure',
Nearest to 'g': 'eight', 'aruban', 'jason', 'africans', 'i', 'am
',
Nearest to 'de': 'publicly', 'oxus', 'prize', 'shortly', 'course
', 'aa',
Nearest to 'a': 'the', 'austin', 'UNK', 'and', 'reginae', 'alpin
a',
Nearest to 'than': 'karts', 'community', 'hesse', 'there', 'raci
al', 'caves',
Average loss at step 12000 is 13.873
Average loss at step 14000 is 11.914
Average loss at step 16000 is 9.810
Average loss at step 18000 is 8.702
Average loss at step 20000 is 7.841
Nearest to 'an': 'the', 'protein', 'overseas', 'their', 'interru
ption', 'counterexample',
Nearest to 'called': 'within', 'gland', 'hiroshima', 'litigants'
, 'dasyprocta', 'clan',
Nearest to 'american': 'and', 'methadone', 'olympics', 'while',
'technologically', 'certainty',
Nearest to 'who': 'and', 'attacked', 'also', 'ruth', 'named', 'a
gouti',
Nearest to 'c': 'eight', 'victoriae', 'accounts', 'one', 'blog',
 'chapman',
Nearest to 'set': 'reginae', 'shortcuts', 'astoria', 'headquarte
rs', 'members', 'few',
Nearest to 'different': 'unreleased', 'mya', 'cl', 'agouti', 'ps
ychologist', 'truetype',
Nearest to 'eight': 'nine', 'six', 'zero', 'five', 'seven', 'thr
ee',
Nearest to 'all': 'bestseller', 'beginning', 'transport', 'waugh
', 'cl', 'lindbergh',
Nearest to 'do': 'condorcet', 'vs', 'proper', 'assistant', 'refl
ect', 'symbols',
Nearest to 'as': 'and', 'for', 'by', 'was', 'is', 'in',
Nearest to 'law': 'agouti', 'threatens', 'collapse', 'clem', 'su
mer', 'soviet',
Nearest to 'UNK': 'agouti', 'alpina', 'victoriae', 'reginae', 'd
asyprocta', 'and',
Nearest to 'by': 'was', 'and', 'in', 'from', 'as', 'with',
Nearest to 'large': 'towers', 'aquila', 'enhancing', 'gland', 'f
ricatives', 'additions',
Nearest to 'their': 'the', 'reginae', 'his', 'afghani', 'a', 'th
is',
Nearest to 'used': 'spiced', 'mystery', 'haliotis', 'mining', 'u
ltraviolet', 'algol',
Nearest to 'british': 'equinox', 'anatomy', 'astor', 'socialism'
, 'victoriae', 'vivian',
Nearest to 'x': 'actaeon', 'envoys', 'four', 'agouti', 'apatosau
rus', 'eight',
Nearest to 'd': 'b', 'and', 'francesco', 'finalist', 'asphyxia',
```

```
 'one',
Nearest to 'king': 'circumscribed', 'inter', 'niacin', 'iphigeni
a', 'four', 'moves',
Nearest to 'based': 'chemist', 'charlie', 'encryption', 'reginae
', 'competitions', 'dim',
Nearest to 'two': 'one', 'three', 'five', 'nine', 'six', 'four',
Nearest to 'international': 'da', 'UNK', 'hiv', 'confidential',
'aloe', 'length',
Nearest to 'him': 'seaport', 'spontaneous', 'infectious', 'trivi
a', 'directors', 'marine',
Nearest to 'year': 'conclude', 'five', 'overwhelming', 'responsi
bility', 'objects', 'subkey',
Nearest to 'in': 'and', 'with', 'of', 'on', 'mya', 'from',
Nearest to 'may': 'sur', 'cannot', 'can', 'seleucid', 'agouti',
'tubing',
Nearest to 'g': 'agouti', 'aruban', 'eight', 'jason', 'i', 'dasy
procta',
Nearest to 'de': 'fita', 'marischal', 'publicly', 'shortly', 'el
ectrophoresis', 'winner',
Nearest to 'a': 'the', 'agouti', 'reginae', 'victoriae', 'and',
'afghani',
Nearest to 'than': 'community', 'karts', 'hesse', 'caves', 'eigh
t', 'ensues',
Average loss at step 22000 is 7.241
```

5. Visualize the embeding

```
def plot_with_labels(low_dim_embs, labels, filename='tsne.png'):
    assert low_dim_embs.shape[0] >= len(labels), "More labels th
an embeddings"
    plt.figure(figsize=(18, 18))  #in inches
    for i, label in enumerate(labels):
        x, y = low_dim_embs[i,:]
        plt.scatter(x, y)
        plt.annotate(label,
                     xy=(x, y),
                     xytext=(5, 2),
                     textcoords='offset points',
                     ha='right',
                     va='bottom')
    plt.show()

# Plot
tsne = TSNE(perplexity=30, n_components=2, init='pca', n_iter=50
00)
plot_only = 500
low_dim_embs = tsne.fit_transform(final_embeddings[:plot_only,:]
)
labels = [reverse_dictionary[i] for i in xrange(plot_only)]
plot_with_labels(low_dim_embs, labels)
```

Auto-Encoder Model

DENOISING AUTO ENCODER

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
%matplotlib inline
print ("PACKAGES LOADED")
```

```
PACKAGES LOADED
```

MNIST

```
mnist = input_data.read_data_sets('data/', one_hot=True)
trainimg   = mnist.train.images
trainlabel = mnist.train.labels
testimg    = mnist.test.images
testlabel  = mnist.test.labels
print ("MNIST LOADED")
```

```
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
MNIST LOADED
```

DEFINE NETWORK

```
# NETOWRK PARAMETERS
n_input    = 784
n_hidden_1 = 256
n_hidden_2 = 256
n_output   = 784

# PLACEHOLDERS
x = tf.placeholder("float", [None, n_input])
y = tf.placeholder("float", [None, n_output])
dropout_keep_prob = tf.placeholder("float")

# WEIGHTS
weights = {
    'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1])),
    'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2])
),
    'out': tf.Variable(tf.random_normal([n_hidden_2, n_output]))
}
biases = {
    'b1': tf.Variable(tf.random_normal([n_hidden_1])),
    'b2': tf.Variable(tf.random_normal([n_hidden_2])),
    'out': tf.Variable(tf.random_normal([n_output]))
}

# MODEL
def dae(_X, _weights, _biases, _keep_prob):
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(_X, _weights['h1'])
, _biases['b1']))
    layer_1out = tf.nn.dropout(layer_1, _keep_prob)
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1out, _weight
s['h2']), _biases['b2']))
    layer_2out = tf.nn.dropout(layer_2, _keep_prob)
    return tf.nn.sigmoid(tf.matmul(layer_2out, _weights['out'])
+ _biases['out'])

# MODEL AS A FUNCTION
recon = dae(x, weights, biases, dropout_keep_prob)
print ("NETOWRK READY")
```

```
NETOWRK READY
```

DEFINE FUNCTIONS

```
# COST
cost = tf.reduce_mean(tf.pow(recon-y, 2))
# OPTIMIZER
optm = tf.train.AdamOptimizer(0.01).minimize(cost)
# INITIALIZER
init = tf.initialize_all_variables()
print ("FUNCTIONS READY")
```

```
FUNCTIONS READY
```

DEFINE SAVER

```
savedir = "nets/"
saver   = tf.train.Saver(max_to_keep=1)
print ("SAVER READY")
```

```
SAVER READY
```

TRAIN

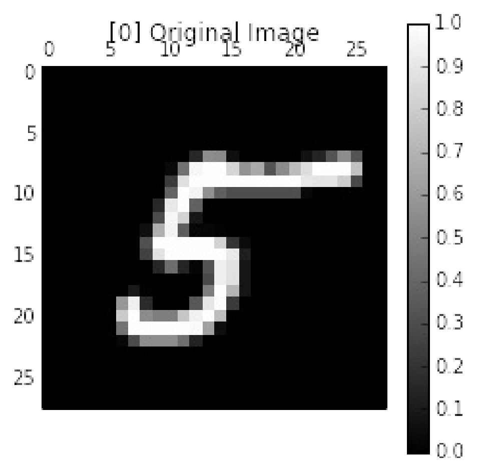
```
TRAIN_FLAG = 1
epochs     = 50
batch_size = 100
disp_step  = 10

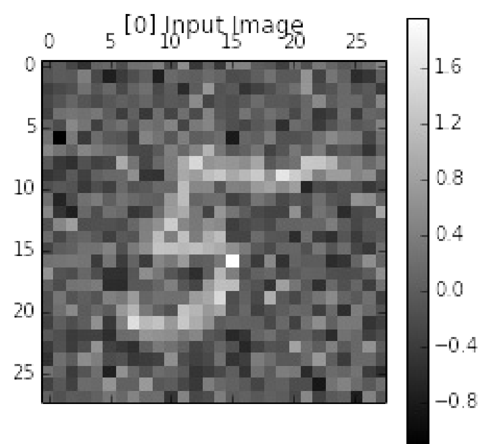
sess = tf.Session()
sess.run(init)
if TRAIN_FLAG:
    print ("START OPTIMIZATION")
    for epoch in range(epochs):
        num_batch  = int(mnist.train.num_examples/batch_size)
        total_cost = 0.
        for i in range(num_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_si
ze)
            batch_xs_noisy = batch_xs + 0.3*np.random.randn(batc
h_size, 784)
            feeds = {x: batch_xs_noisy, y: batch_xs, dropout_kee
p_prob: 1.}
            sess.run(optm, feed_dict=feeds)
            total_cost += sess.run(cost, feed_dict=feeds)
        # DISPLAY
```

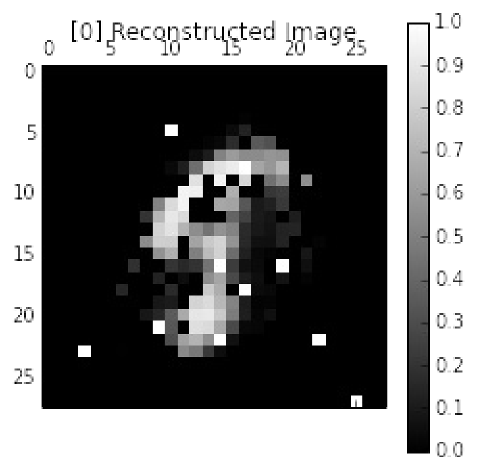
```
        if epoch % disp_step == 0:
            print ("Epoch %02d/%02d average cost: %.6f"
                   % (epoch, epochs, total_cost/num_batch))
            # PLOT
            randidx  = np.random.randint(testimg.shape[0], size=1
)
            testvec  = testimg[randidx, :]
            noisyvec = testvec + 0.3*np.random.randn(1, 784)
            outvec   = sess.run(recon, feed_dict={x: testvec, dr
opout_keep_prob: 1.})
            outimg   = np.reshape(outvec, (28, 28))
            # Plot
            plt.matshow(np.reshape(testvec, (28, 28)), cmap=plt.
get_cmap('gray'))
            plt.title("[" + str(epoch) + "] Original Image")
            plt.colorbar()
            plt.show()
            plt.matshow(np.reshape(noisyvec, (28, 28)), cmap=plt
.get_cmap('gray'))
            plt.title("[" + str(epoch) + "] Input Image")
            plt.colorbar()
            plt.show()
            plt.matshow(outimg, cmap=plt.get_cmap('gray'))
            plt.title("[" + str(epoch) + "] Reconstructed Image"
)
            plt.colorbar()
            plt.show()
        # SAVE
        saver.save(sess, savedir + 'dae.ckpt', global_step=epoch
)
print ("OPTIMIZATION FINISHED")
```

```
START OPTIMIZATION
Epoch 00/50 average cost: 0.085103

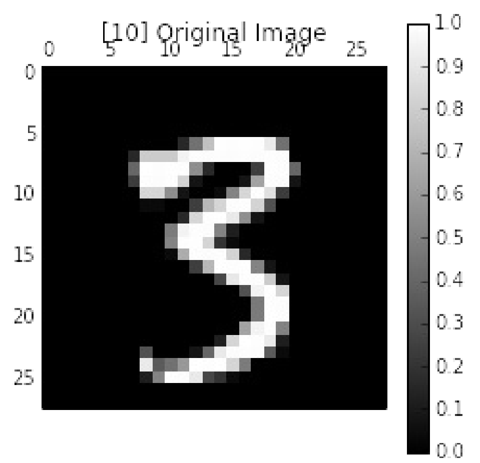

/usr/lib/pymodules/python2.7/matplotlib/collections.py:548: Futu
reWarning: elementwise comparison failed; returning scalar inste
ad, but in the future will perform elementwise comparison
  if self._edgecolors == 'face':
```

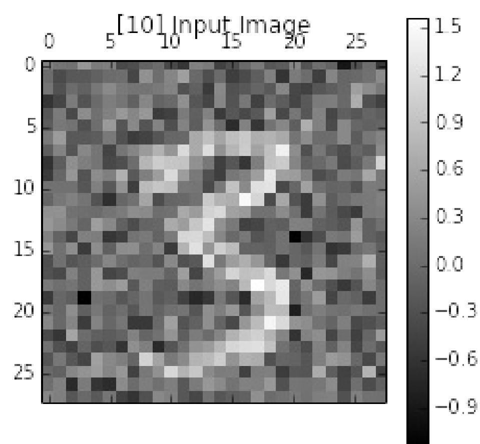


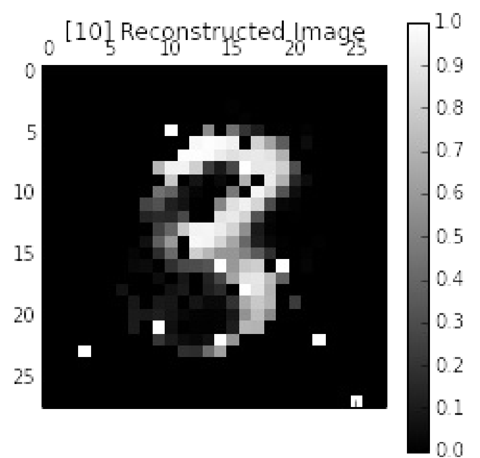




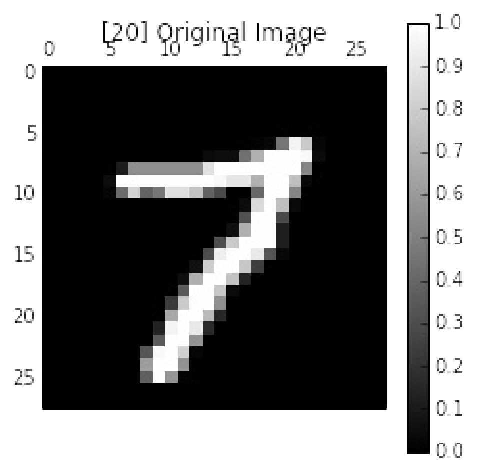

```
Epoch 10/50 average cost: 0.056666
```

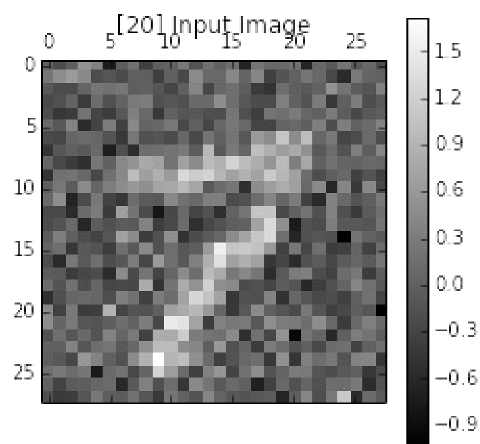


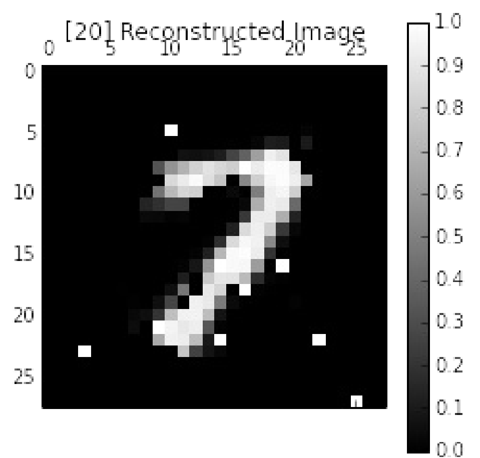




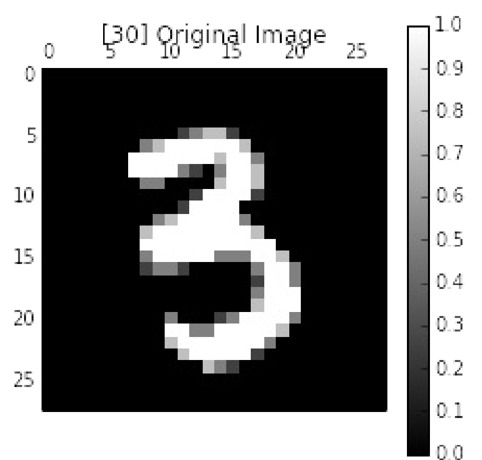

```
Epoch 20/50 average cost: 0.054345
```

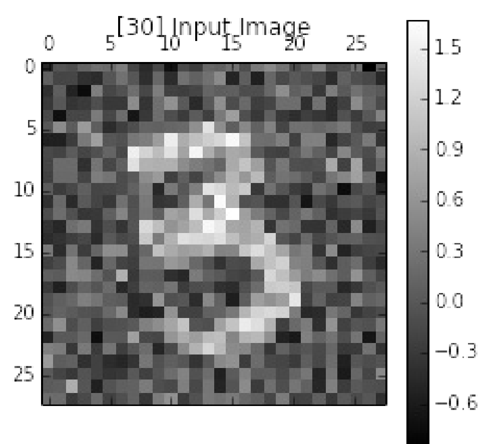


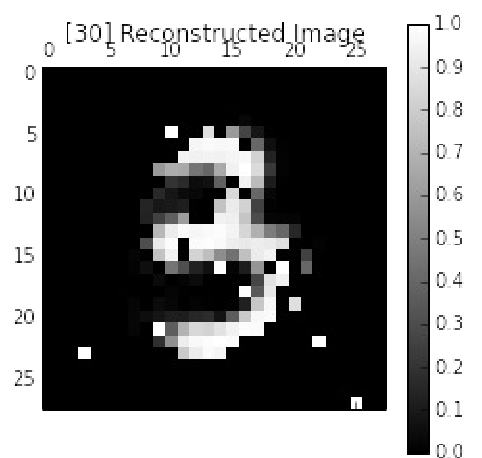




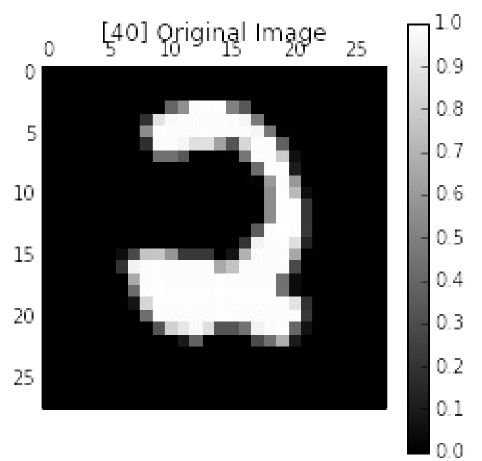

```
Epoch 30/50 average cost: 0.053202
```

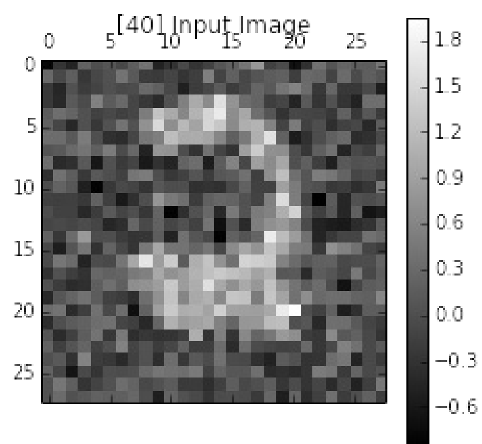


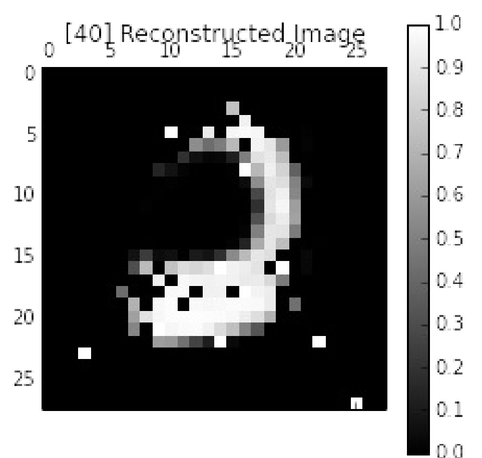





```
Epoch 40/50 average cost: 0.052787
```








```
OPTIMIZATION FINISHED
```

```
# Restore
load_epoch = 40
saver.restore(sess, "nets/dae.ckpt-" + str(load_epoch))
```

```
# Test one
randidx   = np.random.randint(testimg.shape[0], size=1)
orgvec    = testimg[randidx, :]
testvec   = testimg[randidx, :]
label     = np.argmax(testlabel[randidx, :], 1)

print ("label is %d" % (label))
# Noise type
ntype = 2 # 1: Gaussian Noise, 2: Salt and Pepper Noise
if ntype is 1:
    print ("Gaussian Noise")
    noisyvec = testvec + 0.3*np.random.randn(1, 784)
else:
    print ("Salt and Pepper Noise")
    noisyvec = testvec
    rate     = 0.15
    noiseidx = np.random.randint(testimg.shape[1]
                                 , size=int(testimg.shape[1]*rat
e))
    noisyvec[0, noiseidx] = 1-noisyvec[0, noiseidx]

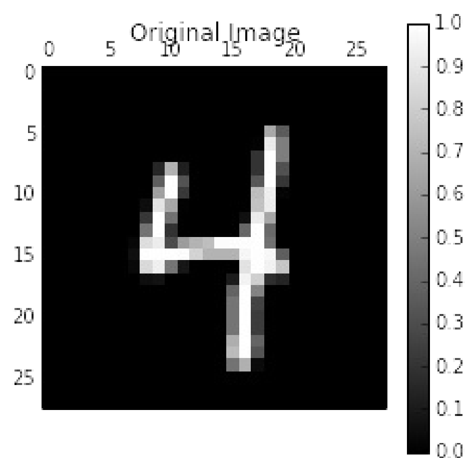
outvec   = sess.run(recon, feed_dict={x: noisyvec, dropout_keep_
prob: 1})
outimg   = np.reshape(outvec, (28, 28))

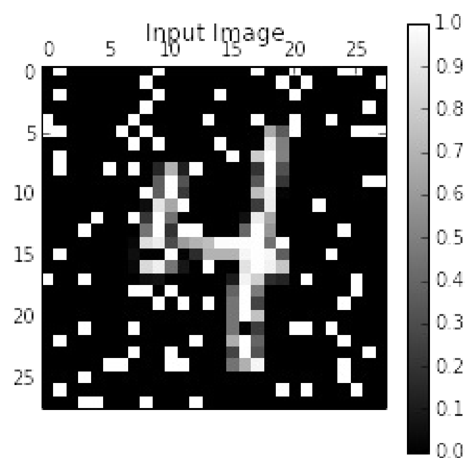
# Plot
plt.matshow(np.reshape(orgvec, (28, 28)), cmap=plt.get_cmap('gra
y'))
plt.title("Original Image")
plt.colorbar()

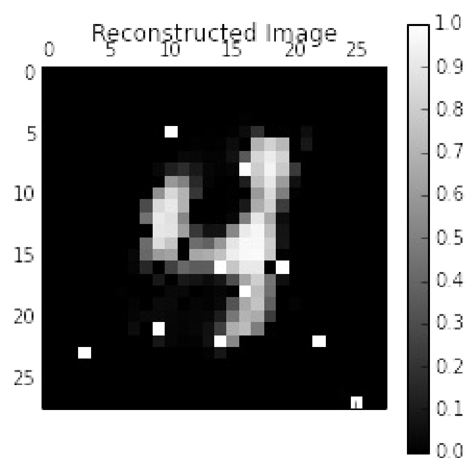
plt.matshow(np.reshape(noisyvec, (28, 28)), cmap=plt.get_cmap('g
ray'))
plt.title("Input Image")
plt.colorbar()

plt.matshow(outimg, cmap=plt.get_cmap('gray'))
plt.title("Reconstructed Image")
plt.colorbar()
plt.show()
```

```
label is 4
Salt and Pepper Noise
```








```
# Visualize Filter
from PIL import Image

def scale_to_unit_interval(ndar, eps=1e-8):
    """ Scales all values in the ndarray ndar to be between 0 an
d 1 """
    ndar = ndar.copy()
    ndar -= ndar.min()
    ndar *= 1.0 / (ndar.max() + eps)
    return ndar
```

```
def tile_raster_images(X, img_shape, tile_shape, tile_spacing=(0
, 0),
                       scale_rows_to_unit_interval=True,
                       output_pixel_vals=True):
    assert len(img_shape) == 2
    assert len(tile_shape) == 2
    assert len(tile_spacing) == 2
    out_shape = [(ishp + tsp) * tshp - tsp for ishp, tshp, tsp
                      in zip(img_shape, tile_shape, tile_spacing
)]

    if isinstance(X, tuple):
        assert len(X) == 4
        # Create an output numpy ndarray to store the image
        if output_pixel_vals:
            out_array = np.zeros((out_shape[0], out_shape[1], 4)
, dtype='uint8')
        else:
            out_array = np.zeros((out_shape[0], out_shape[1], 4)
, dtype=X.dtype)

        #colors default to 0, alpha defaults to 1 (opaque)
        if output_pixel_vals:
            channel_defaults = [0, 0, 0, 255]
        else:
            channel_defaults = [0., 0., 0., 1.]

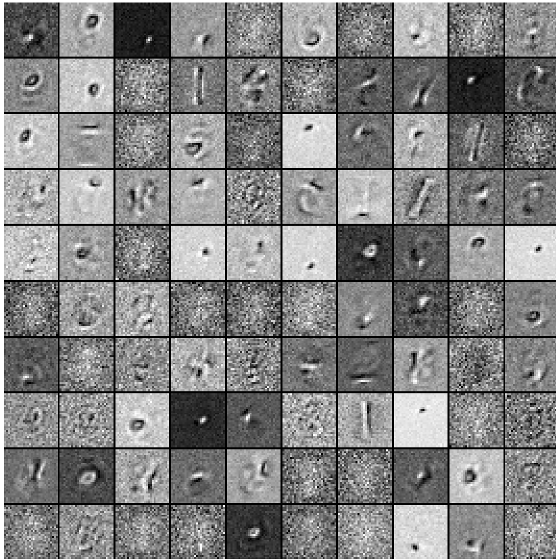
        for i in range(4):
            if X[i] is None:
                # if channel is None, fill it with zeros of the
correct
                # dtype
                out_array[:, :, i] = np.zeros(out_shape,
                      dtype='uint8' if output_pixel_vals else ou
t_array.dtype
                      ) + channel_defaults[i]
            else:
                # use a recurrent call to compute the channel an
d store it
                # in the output
                out_array[:, :, i] = tile_raster_images(X[i], im
g_shape, tile_shape, tile_spacing, scale_rows_to_unit_interval,
output_pixel_vals)
        return out_array

    else:
        # if we are dealing with only one channel
        H, W = img_shape
        Hs, Ws = tile_spacing

        # generate a matrix to store the output
        out_array = np.zeros(out_shape, dtype='uint8' if output_
pixel_vals else X.dtype)
```

```
        for tile_row in range(tile_shape[0]):
            for tile_col in range(tile_shape[1]):
                if tile_row * tile_shape[1] + tile_col < X.shape[
0]:
                    if scale_rows_to_unit_interval:
                        # if we should scale values to be betwee
n 0 and 1
                        # do this by calling the `scale_to_unit_
interval`
                        # function
                        this_img = scale_to_unit_interval(X[tile
_row * tile_shape[1] + tile_col].reshape(img_shape))
                    else:
                        this_img = X[tile_row * tile_shape[1] +
tile_col].reshape(img_shape)
                    # add the slice to the corresponding positio
n in the
                    # output array
                    out_array[
                        tile_row * (H+Hs): tile_row * (H + Hs) +
 H,
                        tile_col * (W+Ws): tile_col * (W + Ws) +
 W
                        ] \
                        = this_img * (255 if output_pixel_vals e
lse 1)
        return out_array

# Visualize filter
w1 = sess.run(weights["h1"])
image = Image.fromarray(tile_raster_images(
        X = w1.T,
        img_shape=(28, 28), tile_shape=(10, 10),
        tile_spacing=(1, 1)))
image
```



DENOISING AUTO-ENCODER

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
%matplotlib inline
print ("PACKAGES LOADED")
```

```
PACKAGES LOADED
```

MNIST

```
mnist = input_data.read_data_sets('data/', one_hot=True)
trainimg   = mnist.train.images
trainlabel = mnist.train.labels
testimg    = mnist.test.images
testlabel  = mnist.test.labels
```

```
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
```

DEVICE TO USE

```
device2use = "/gpu:0"
```

DEFINE NETWORK

```
# Network Parameters
n_input    = 784 # MNIST data input (img shape: 28*28)
n_hidden_1 = 256 # 1st layer num features
n_hidden_2 = 256 # 2nd layer num features
n_output   = 784 #
with tf.device(device2use):
    # tf Graph input
    x = tf.placeholder("float", [None, n_input])
    y = tf.placeholder("float", [None, n_output])
    dropout_keep_prob = tf.placeholder("float")
    # Store layers weight & bias
    weights = {
        'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1]
)),
        'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden
_2])),
        'out': tf.Variable(tf.random_normal([n_hidden_2, n_outpu
t]))
    }
    biases = {
        'b1': tf.Variable(tf.random_normal([n_hidden_1])),
        'b2': tf.Variable(tf.random_normal([n_hidden_2])),
        'out': tf.Variable(tf.random_normal([n_output]))
    }
```

```
with tf.device(device2use):
    # Create model
    def denoising_autoencoder(_X, _weights, _biases, _keep_prob):

        layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(_X, _weights['h
1']), _biases['b1']))
        layer_1out = tf.nn.dropout(layer_1, _keep_prob)
        layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1out, _we
ights['h2']), _biases['b2']))
        layer_2out = tf.nn.dropout(layer_2, _keep_prob)
        return tf.nn.sigmoid(tf.matmul(layer_2out, _weights['out'
]) + _biases['out'])
print ("NETWORK READY")
```

```
NETWORK READY
```

DEFINE FUNCTIONS

```
with tf.device(device2use):
    # MODEL
    out = denoising_autoencoder(x, weights, biases, dropout_keep
_prob)
    # DEFINE LOSS AND OPTIMIZER
    cost = tf.reduce_mean(tf.pow(out-y, 2))
    optimizer = tf.train.AdamOptimizer(learning_rate=0.01).minim
ize(cost)
    # INITIALIZE
    init = tf.initialize_all_variables()
    # SAVER
    savedir = "nets/"
    saver = tf.train.Saver(max_to_keep=3)
print ("FUNCTIONS READY")
```

```
FUNCTIONS READY
```

OPTIMIZE

```
do_train = 1
sess = tf.Session(config=tf.ConfigProto(
        allow_soft_placement=True, log_device_placement=True))
sess.run(init)
```

```
training_epochs = 30
batch_size      = 100
display_step    = 5
plot_step       = 10
if do_train:
    print ("START OPTIMIZATION")
    for epoch in range(training_epochs):
        avg_cost = 0.
        num_batch = int(mnist.train.num_examples/batch_size)
        for i in range(num_batch):
            randidx = np.random.randint(trainimg.shape[0], size=
batch_size)
            batch_xs = trainimg[randidx, :]
            batch_xs_noisy = batch_xs + 0.3*np.random.randn(batc
h_xs.shape[0], 784)
            feed1 = {x: batch_xs_noisy, y: batch_xs, dropout_kee
p_prob: 0.5}
            sess.run(optimizer, feed_dict=feed1)
            feed2 = {x: batch_xs_noisy, y: batch_xs, dropout_kee
p_prob: 1}
            avg_cost += sess.run(cost, feed_dict=feed2)/num_batc
```

```
h

        # DISPLAY
        if epoch % display_step == 0:
            print ("Epoch: %03d/%03d cost: %.9f" % (epoch, train
ing_epochs, avg_cost))
        if epoch % plot_step == 0 or epoch == training_epochs-1:
            # TEST
            randidx  = np.random.randint(testimg.shape[0], size=1
)
            testvec  = testimg[randidx, :]
            noisyvec = testvec + 0.3*np.random.randn(1, 784)
            outvec   = sess.run(out, feed_dict={x: testvec, drop
out_keep_prob: 1.})
            outimg   = np.reshape(outvec, (28, 28))

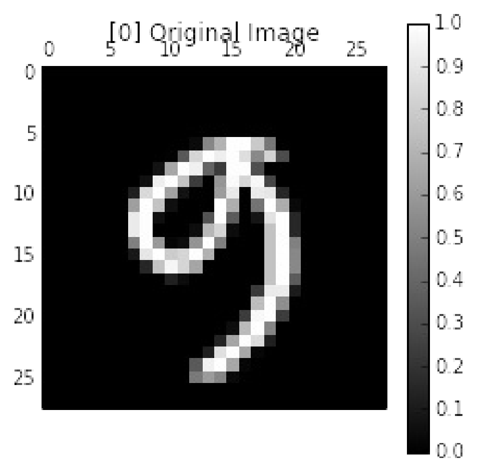
            # PLOT
            plt.matshow(np.reshape(testvec, (28, 28)), cmap=plt.
get_cmap('gray'))
            plt.title("[" + str(epoch) + "] Original Image")
            plt.colorbar()
            plt.matshow(np.reshape(noisyvec, (28, 28)), cmap=plt
.get_cmap('gray'))
            plt.title("[" + str(epoch) + "] Input Image")
            plt.colorbar()
            plt.matshow(outimg, cmap=plt.get_cmap('gray'))
            plt.title("[" + str(epoch) + "] Reconstructed Image"
)
            plt.colorbar()
            plt.show()

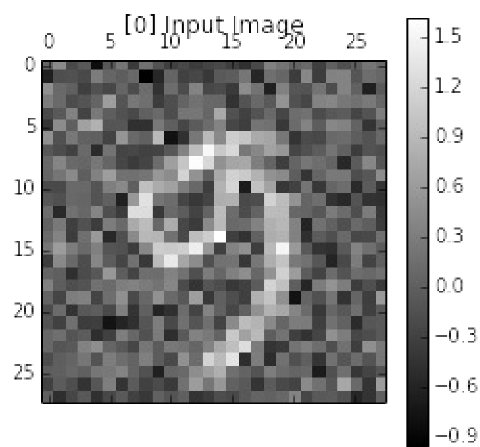
            # SAVE
            saver.save(sess, savedir + 'dae_dr.ckpt', global_ste
p=epoch)

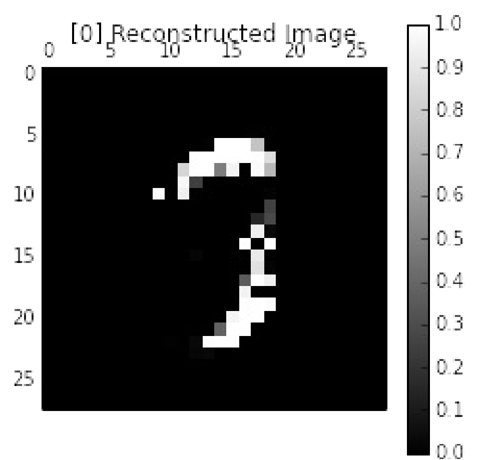
    print ("Optimization Finished!")
else:
    print ("RESTORE")
    saver.restore(sess, "nets/dae_dr.ckpt-" + str(training_epochs
-1))
```

```
START OPTIMIZATION
Epoch: 000/030 cost: 0.107168743

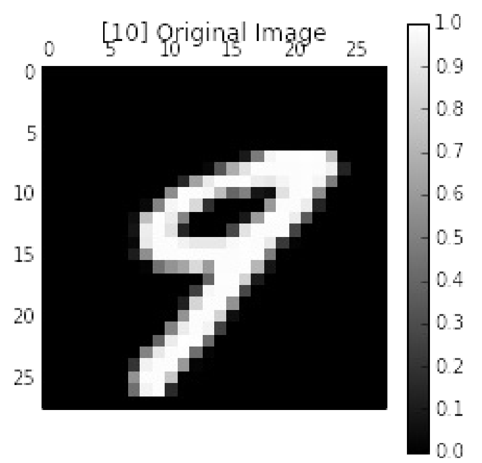

/usr/lib/pymodules/python2.7/matplotlib/collections.py:548: Futu
reWarning: elementwise comparison failed; returning scalar inste
ad, but in the future will perform elementwise comparison
  if self._edgecolors == 'face':
```

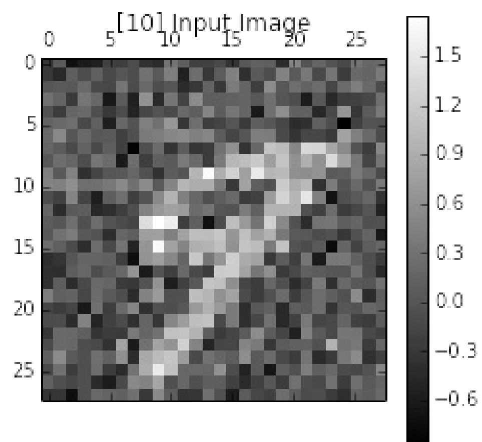


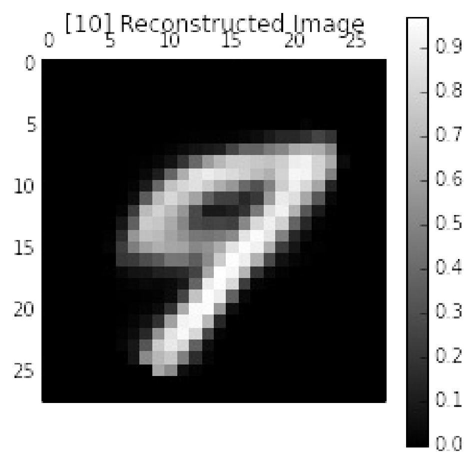




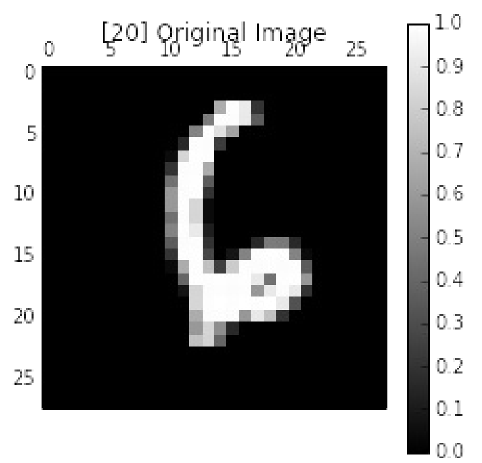

```
Epoch: 005/030 cost: 0.033176764
Epoch: 010/030 cost: 0.026914674
```

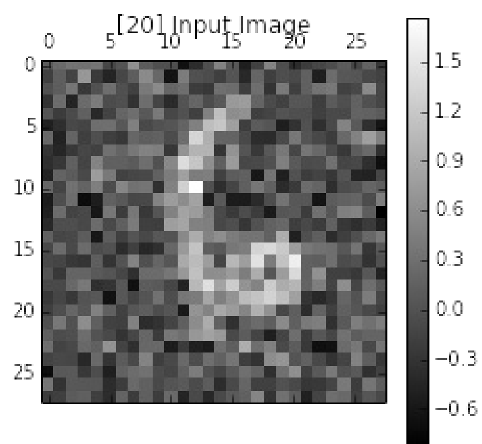


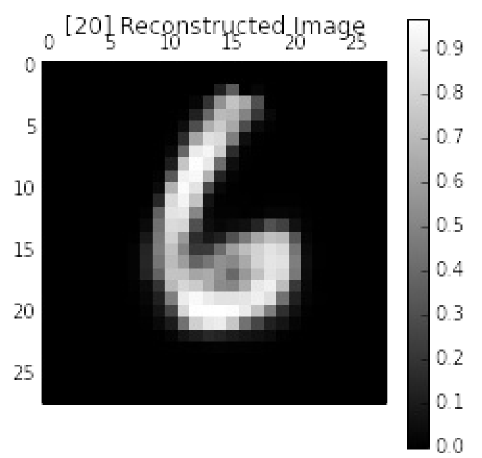




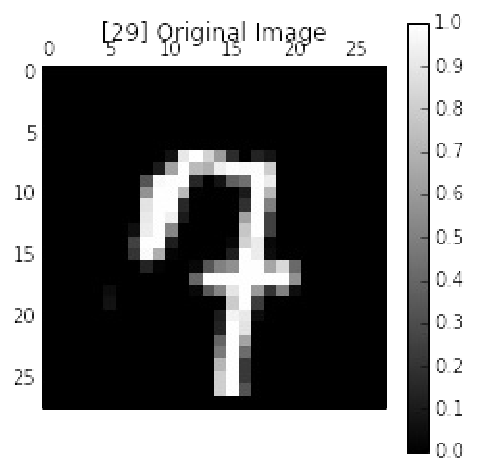

```
Epoch: 015/030 cost: 0.024993547
Epoch: 020/030 cost: 0.023792257
```

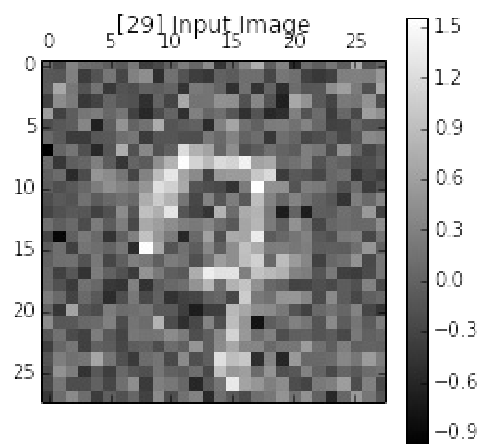


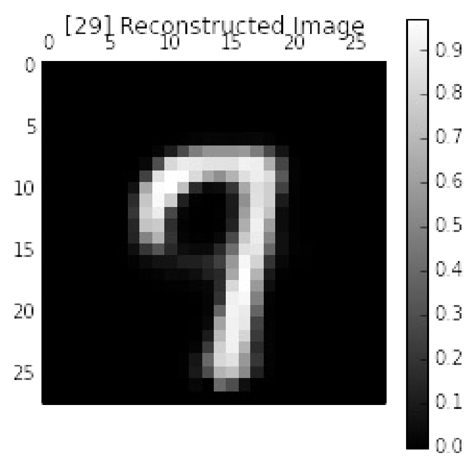





```
Epoch: 025/030 cost: 0.023201655
```








```
Optimization Finished!
```

TEST

```
randidx   = np.random.randint(testimg.shape[0], size=1)
orgvec    = testimg[randidx, :]
testvec   = testimg[randidx, :]
label     = np.argmax(testlabel[randidx, :], 1)

print ("label is %d" % (label))
# Noise type
ntype = 2 # 1: Gaussian Noise, 2: Salt and Pepper Noise
if ntype is 1:
    print ("Gaussian Noise")
    noisyvec = testvec + 0.1*np.random.randn(1, 784)
else:
    print ("Salt and Pepper Noise")
    noisyvec = testvec
    rate     = 0.20
    noiseidx = np.random.randint(testimg.shape[1], size=int(test
img.shape[1]*rate))
    noisyvec[0, noiseidx] = 1-noisyvec[0, noiseidx]

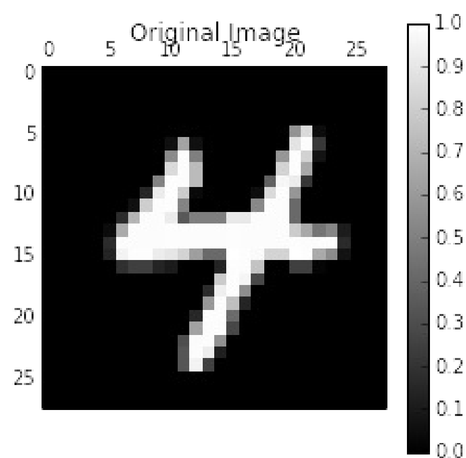
outvec   = sess.run(out, feed_dict={x: noisyvec, dropout_keep_pr
ob: 1})
outimg   = np.reshape(outvec, (28, 28))

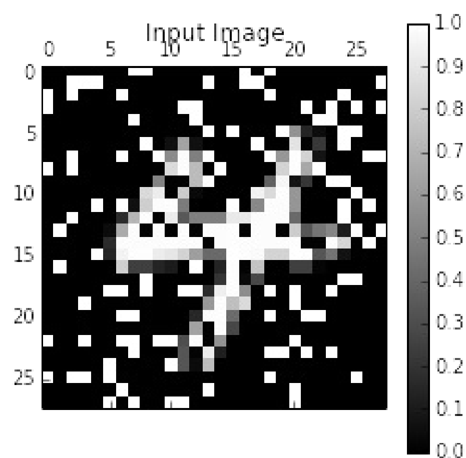
# Plot
plt.matshow(np.reshape(orgvec, (28, 28)), cmap=plt.get_cmap('gra
y'))
plt.title("Original Image")
plt.colorbar()

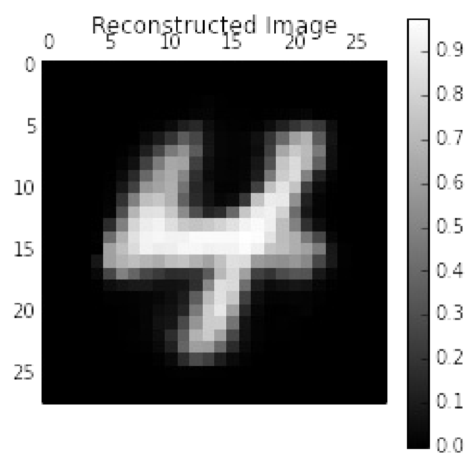
plt.matshow(np.reshape(noisyvec, (28, 28)), cmap=plt.get_cmap('g
ray'))
plt.title("Input Image")
plt.colorbar()

plt.matshow(outimg, cmap=plt.get_cmap('gray'))
plt.title("Reconstructed Image")
plt.colorbar()
plt.show()
```

```
label is 4
Salt and Pepper Noise
```







PLOT FILTER SHAPES

```
# Visualize Filter
from PIL import Image

def scale_to_unit_interval(ndar, eps=1e-8):
    """ Scales all values in the ndarray ndar to be between 0 an
d 1 """
    ndar = ndar.copy()
```

```
    ndar -= ndar.min()
    ndar *= 1.0 / (ndar.max() + eps)
    return ndar

def tile_raster_images(X, img_shape, tile_shape, tile_spacing=(0
, 0),
                       scale_rows_to_unit_interval=True,
                       output_pixel_vals=True):
    assert len(img_shape) == 2
    assert len(tile_shape) == 2
    assert len(tile_spacing) == 2
    out_shape = [(ishp + tsp) * tshp - tsp for ishp, tshp, tsp
                      in zip(img_shape, tile_shape, tile_spacing
)]

    if isinstance(X, tuple):
        assert len(X) == 4
        # Create an output numpy ndarray to store the image
        if output_pixel_vals:
            out_array = np.zeros((out_shape[0], out_shape[1], 4)
, dtype='uint8')
        else:
            out_array = np.zeros((out_shape[0], out_shape[1], 4)
, dtype=X.dtype)

        #colors default to 0, alpha defaults to 1 (opaque)
        if output_pixel_vals:
            channel_defaults = [0, 0, 0, 255]
        else:
            channel_defaults = [0., 0., 0., 1.]

        for i in range(4):
            if X[i] is None:
                # if channel is None, fill it with zeros of the
correct
                # dtype
                out_array[:, :, i] = np.zeros(out_shape,
                      dtype='uint8' if output_pixel_vals else ou
t_array.dtype
                      ) + channel_defaults[i]
            else:
                # use a recurrent call to compute the channel an
d store it
                # in the output
                out_array[:, :, i] = tile_raster_images(X[i], im
g_shape, tile_shape, tile_spacing, scale_rows_to_unit_interval,
output_pixel_vals)
        return out_array

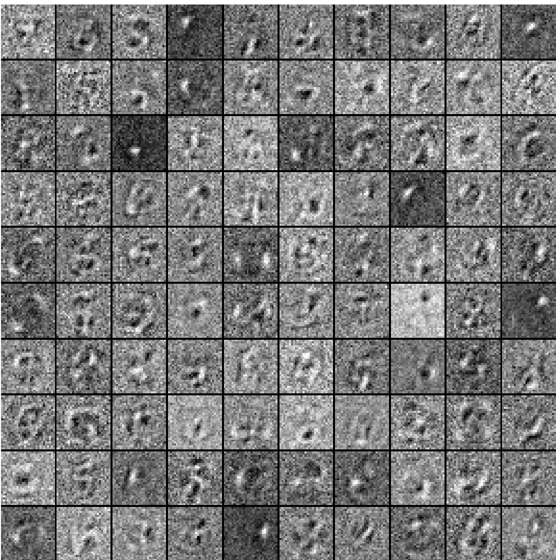
    else:
        # if we are dealing with only one channel
        H, W = img_shape
        Hs, Ws = tile_spacing
```

```
        # generate a matrix to store the output
        out_array = np.zeros(out_shape, dtype='uint8' if output_
pixel_vals else X.dtype)


        for tile_row in range(tile_shape[0]):
            for tile_col in range(tile_shape[1]):
                if tile_row * tile_shape[1] + tile_col < X.shape[
0]:
                    if scale_rows_to_unit_interval:
                        # if we should scale values to be betwee
n 0 and 1
                        # do this by calling the `scale_to_unit_
interval`
                        # function
                        this_img = scale_to_unit_interval(X[tile
_row * tile_shape[1] + tile_col].reshape(img_shape))
                    else:
                        this_img = X[tile_row * tile_shape[1] +
tile_col].reshape(img_shape)
                    # add the slice to the corresponding positio
n in the
                    # output array
                    out_array[
                        tile_row * (H+Hs): tile_row * (H + Hs) +
 H,
                        tile_col * (W+Ws): tile_col * (W + Ws) +
 W
                        ] \
                        = this_img * (255 if output_pixel_vals e
lse 1)
        return out_array

# Visualize filter
w1 = sess.run(weights["h1"])

image = Image.fromarray(tile_raster_images(
        X = w1.T,
        img_shape=(28, 28), tile_shape=(10, 10),
        tile_spacing=(1, 1)))
image
```



Convolutional auto-encoder

```
import matplotlib.pyplot as plt
import numpy as np
import math
import tensorflow as tf
import tensorflow.examples.tutorials.mnist.input_data as input_d
ata
%matplotlib inline

mnist = input_data.read_data_sets("data/", one_hot=True)
trainimgs   = mnist.train.images
trainlabels = mnist.train.labels
testimgs    = mnist.test.images
testlabels  = mnist.test.labels
ntrain      = trainimgs.shape[0]
ntest       = testimgs.shape[0]
dim         = trainimgs.shape[1]
nout        = trainlabels.shape[1]
print ("Packages loaded")
```

```
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
Packages loaded
```

DEFINE NETWORKS

```
n1 = 16
n2 = 32
n3 = 64
ksize = 5
weights = {
    'ce1': tf.Variable(tf.random_normal([ksize, ksize, 1, n1],
stddev=0.1)),
    'ce2': tf.Variable(tf.random_normal([ksize, ksize, n1, n2],
stddev=0.1)),
    'ce3': tf.Variable(tf.random_normal([ksize, ksize, n2, n3],
stddev=0.1)),
    'cd3': tf.Variable(tf.random_normal([ksize, ksize, n2, n3],
stddev=0.1)),
    'cd2': tf.Variable(tf.random_normal([ksize, ksize, n1, n2],
stddev=0.1)),
    'cd1': tf.Variable(tf.random_normal([ksize, ksize, 1, n1],
stddev=0.1))
}
biases = {
    'be1': tf.Variable(tf.random_normal([n1], stddev=0.1)),
    'be2': tf.Variable(tf.random_normal([n2], stddev=0.1)),
    'be3': tf.Variable(tf.random_normal([n3], stddev=0.1)),
    'bd3': tf.Variable(tf.random_normal([n2], stddev=0.1)),
    'bd2': tf.Variable(tf.random_normal([n1], stddev=0.1)),
    'bd1': tf.Variable(tf.random_normal([1],  stddev=0.1))
}
```

```
Network ready
```

```
def cae(_X, _W, _b, _keepprob):
    _input_r = tf.reshape(_X, shape=[-1, 28, 28, 1])
    # Encoder
    _ce1 = tf.nn.sigmoid(tf.add(tf.nn.conv2d(_input_r, _W['ce1']
        , strides=[1, 2, 2, 1], padding='SAME'), _b['be1']))
    _ce1 = tf.nn.dropout(_ce1, _keepprob)
    _ce2 = tf.nn.sigmoid(tf.add(tf.nn.conv2d(_ce1, _W['ce2']
        , strides=[1, 2, 2, 1], padding='SAME'), _b['be2']))
    _ce2 = tf.nn.dropout(_ce2, _keepprob)
    _ce3 = tf.nn.sigmoid(tf.add(tf.nn.conv2d(_ce2, _W['ce3']
        , strides=[1, 2, 2, 1], padding='SAME'), _b['be3']))
    _ce1 = tf.nn.dropout(_ce3, _keepprob)
    # Decoder
    _cd3 = tf.nn.sigmoid(tf.add(tf.nn.conv2d_transpose(_ce3, _W[
'cd3']
        , tf.pack([tf.shape(_X)[0], 7, 7, n2]), strides=[1, 2, 2
, 1]
        , padding='SAME'), _b['bd3']))
    _cd3 = tf.nn.dropout(_cd3, _keepprob)
    _cd2 = tf.nn.sigmoid(tf.add(tf.nn.conv2d_transpose(_cd3, _W[
'cd2']
        , tf.pack([tf.shape(_X)[0], 14, 14, n1]), strides=[1, 2,
2, 1]
        , padding='SAME') , _b['bd2']))
    _cd2 = tf.nn.dropout(_cd2, _keepprob)
    _cd1 = tf.nn.sigmoid(tf.add(tf.nn.conv2d_transpose(_cd2, _W[
'cd1']
        , tf.pack([tf.shape(_X)[0], 28, 28, 1]), strides=[1, 2, 2
, 1]
        , padding='SAME'), _b['bd1']))
    _cd1 = tf.nn.dropout(_cd1, _keepprob)
    _out = _cd1
    return {'input_r': _input_r, 'ce1': _ce1, 'ce2': _ce2, 'ce3'
: _ce3
        , 'cd3': _cd3, 'cd2': _cd2, 'cd1': _cd1
        , 'layers': (_input_r, _ce1, _ce2, _ce3, _cd3, _cd2, _cd
1)
        , 'out': _out}
print ("Network ready")
```

DEFINE FUNCTIONS

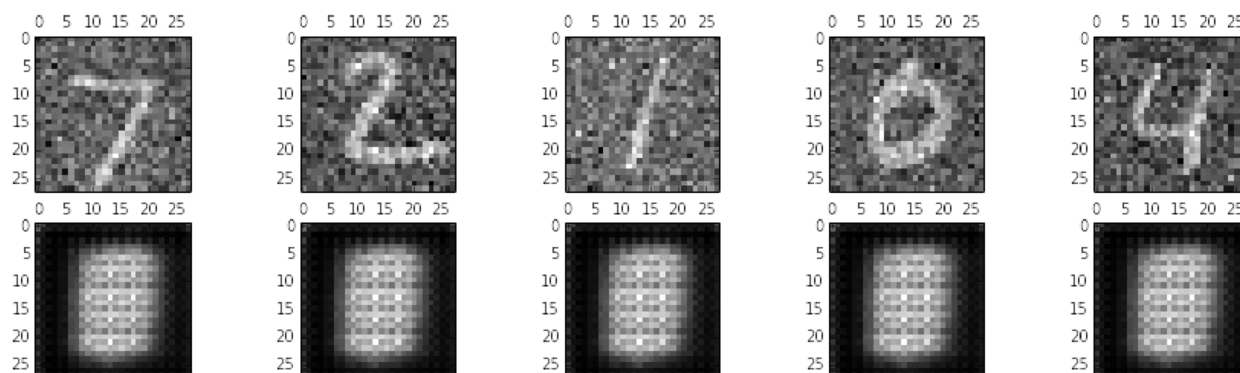
```
x = tf.placeholder(tf.float32, [None, dim])
y = tf.placeholder(tf.float32, [None, dim])
keepprob = tf.placeholder(tf.float32)
pred = cae(x, weights, biases, keepprob)['out']
cost = tf.reduce_sum(tf.square(cae(x, weights, biases, keepprob)[
'out']
            - tf.reshape(y, shape=[-1, 28, 28, 1])))
learning_rate = 0.001
optm = tf.train.AdamOptimizer(learning_rate).minimize(cost)
init = tf.initialize_all_variables()
print ("Functions ready")
```

```
Functions ready
```

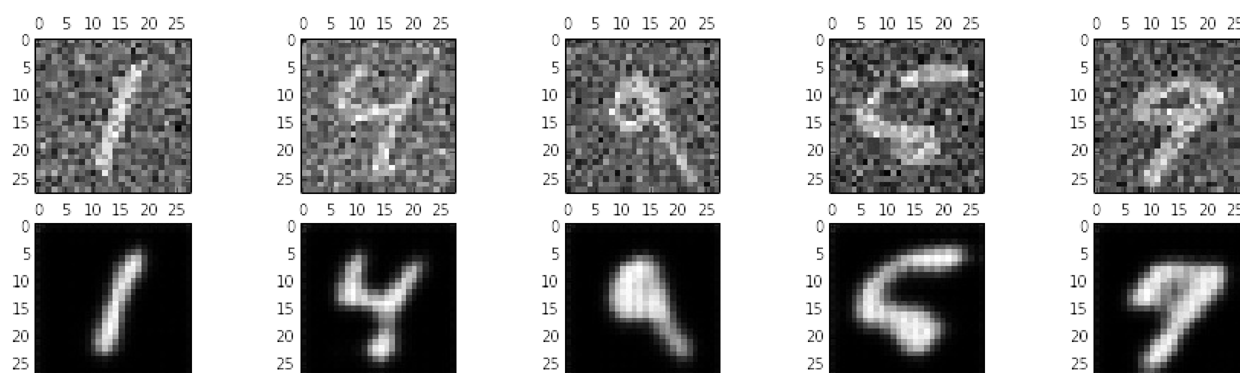
OPTIMIZE

```
sess = tf.Session()
sess.run(init)
# mean_img = np.mean(mnist.train.images, axis=0)
mean_img = np.zeros((784))
# Fit all training data
batch_size = 128
n_epochs   = 5
print("Strart training..")
for epoch_i in range(n_epochs):
    for batch_i in range(mnist.train.num_examples // batch_size)
:
        batch_xs, _ = mnist.train.next_batch(batch_size)
        trainbatch = np.array([img - mean_img for img in batch_x
s])
        trainbatch_noisy = trainbatch + 0.3*np.random.randn(
            trainbatch.shape[0], 784)
        sess.run(optm, feed_dict={x: trainbatch_noisy
                                 , y: trainbatch, keepprob: 0.7
})
    print ("[%02d/%02d] cost: %.4f" % (epoch_i, n_epochs
        , sess.run(cost, feed_dict={x: trainbatch_noisy
                                   , y: trainbatch, keepprob: 1.
})))
    if (epoch_i % 1) == 0:
        n_examples = 5
        test_xs, _ = mnist.test.next_batch(n_examples)
        test_xs_noisy = test_xs + 0.3*np.random.randn(
            test_xs.shape[0], 784)
        recon = sess.run(pred, feed_dict={x: test_xs_noisy, keep
prob: 1.})
        fig, axs = plt.subplots(2, n_examples, figsize=(15, 4))
        for example_i in range(n_examples):
            axs[0][example_i].matshow(np.reshape(
                test_xs_noisy[example_i, :], (28, 28))
                , cmap=plt.get_cmap('gray'))
            axs[1][example_i].matshow(np.reshape(
                np.reshape(recon[example_i, ...], (784,))
                + mean_img, (28, 28)), cmap=plt.get_cmap('gray')
)
        plt.show()
print("Training done. ")
```

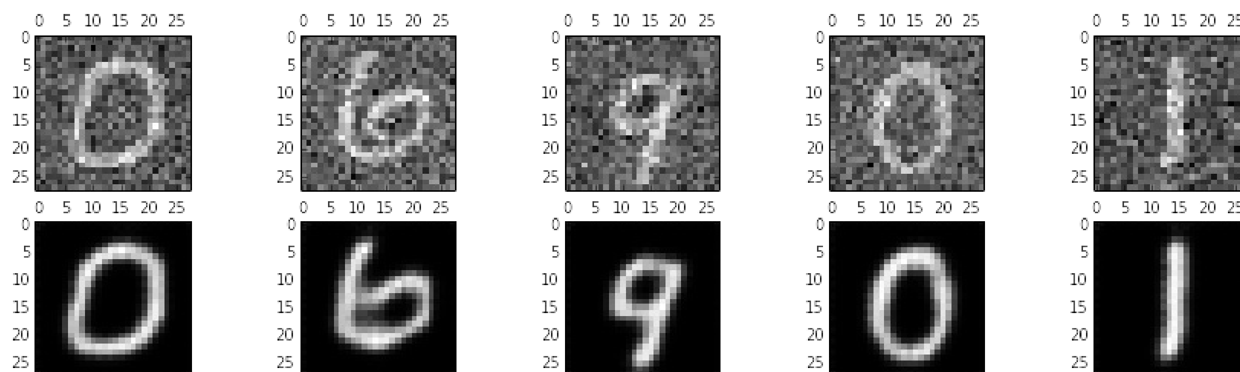
```
Strart training..
[00/05] cost: 8200.5146
```



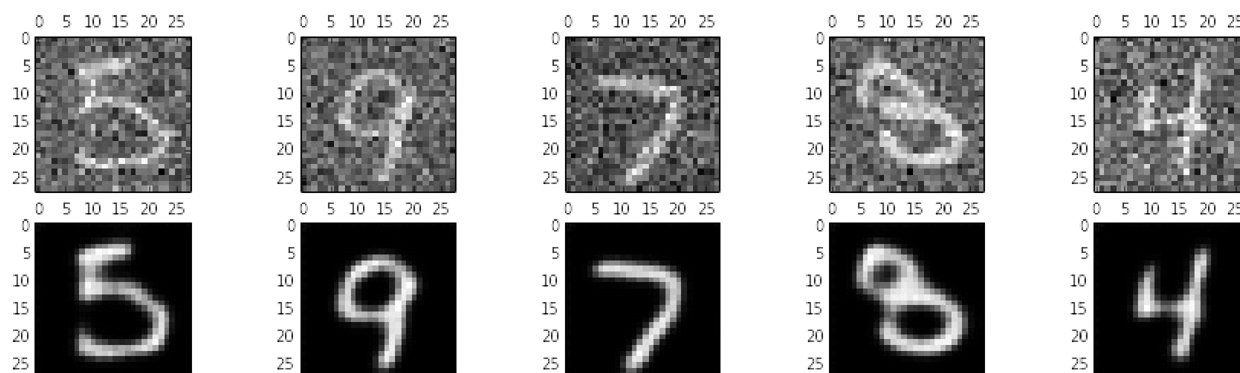
```
[01/05] cost: 3641.4082
```



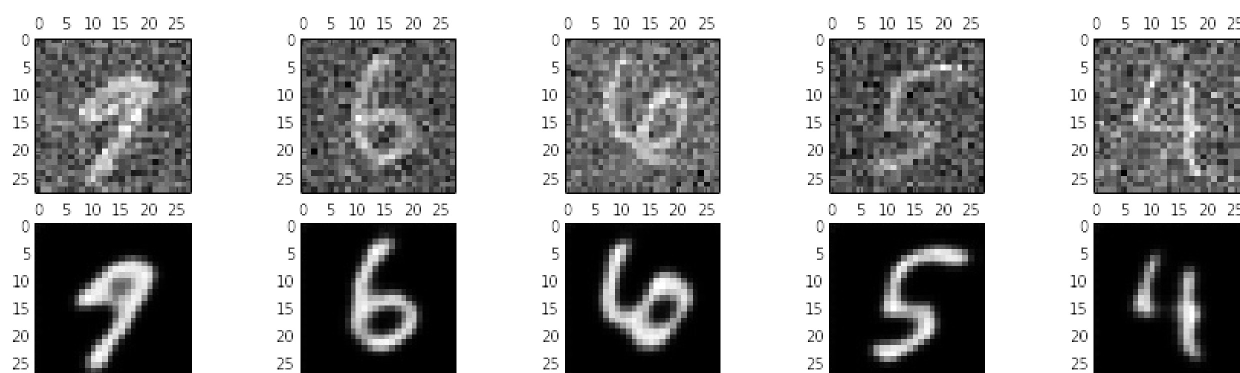
```
[02/05] cost: 2731.7280
```



```
[03/05] cost: 2320.9961
```



```
[04/05] cost: 2219.6125
```



```
Training done.
```

PRINT SHAPE OF THE NETWORK

```
test_xs, _ = mnist.test.next_batch(1)
test_xs_norm = np.array([img - mean_img for img in test_xs])
recon = sess.run(pred, feed_dict={x: test_xs_norm, keepprob: 1.}
)
layers = sess.run(cae(x, weights, biases, keepprob)['layers']
                  , feed_dict={x: test_xs_norm, keepprob: 1.})
for i in range(len(layers)):
    currl = layers[i]
    print (("Shape of layer %d is %s") % (i+1, currl.shape,))
```

```
Shape of layer 1 is (1, 28, 28, 1)
Shape of layer 2 is (1, 4, 4, 64)
Shape of layer 3 is (1, 7, 7, 32)
Shape of layer 4 is (1, 4, 4, 64)
Shape of layer 5 is (1, 7, 7, 32)
Shape of layer 6 is (1, 14, 14, 16)
Shape of layer 7 is (1, 28, 28, 1)
```

CLOSE SESSION

```
sess.close()
print ("Session closed.")
```

```
Session closed.
```

Class Activation Map (CAM)

```
"""
 Weakly Supervised Net (Global Average Pooling) with MNIST
 @Sungjoon Choi (sungjoon.choi@cpslab.snu.ac.kr
"""
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
%matplotlib inline

mnist = input_data.read_data_sets('data/', one_hot=True)
trainimgs   = mnist.train.images
trainlabels = mnist.train.labels
testimgs    = mnist.test.images
testlabels  = mnist.test.labels

ntrain      = trainimgs.shape[0]
ntest       = testimgs.shape[0]
dim         = trainimgs.shape[1]
nout        = trainlabels.shape[1]
```

```
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
```

```
# Make multilayer perceptron

weights = {
    'wc1' : tf.Variable(tf.random_normal([3, 3, 1, 128], stddev=
0.1)),
    'wc2' : tf.Variable(tf.random_normal([3, 3, 128, 256], stdde
v=0.1)),
    'out' : tf.Variable(tf.random_normal([256, 10], stddev=0.1))
}
biases = {
    'bc1' : tf.Variable(tf.random_normal([128], stddev=0.1)),
    'bc2' : tf.Variable(tf.random_normal([256], stddev=0.1)),
    'out' : tf.Variable(tf.random_normal([10], stddev=0.1))
}

def mlp(_X, _W, _b, _keepprob):
    # Reshape input
    _input_r = tf.reshape(_X, shape=[-1, 28, 28, 1])
    # Conv1
    _conv1 = tf.nn.relu(
            tf.nn.bias_add(
                tf.nn.conv2d(_input_r, _W['wc1'], strides=[1, 1,
1, 1], padding='SAME')
```

```
            , _b['bc1'])
        )
    # Pool1
    _pool1 = tf.nn.max_pool(_conv1, ksize=[1, 2, 2, 1], strides=[
1, 2, 2, 1], padding='SAME')
    # DropOut1
    _layer1 = tf.nn.dropout(_pool1, _keepprob)
    # Conv2
    _conv2 = tf.nn.relu(
            tf.nn.bias_add(
                tf.nn.conv2d(_layer1, _W['wc2'], strides=[1, 1, 1
, 1], padding='SAME')
            , _b['bc2'])
        )
    # Pool2 (Global average pooling)
    _pool2 = tf.nn.avg_pool(_conv2, ksize=[1, 14, 14, 1], stride
s=[1, 14, 14, 1], padding='SAME')
    # DropOut2
    _layer2 = tf.nn.dropout(_pool2, _keepprob)
    # Vectorize
    _dense = tf.reshape(_layer2, [-1, _W['out'].get_shape().as_l
ist()[0]])
    # FC1
    _out   = tf.nn.softmax(tf.add(tf.matmul(_dense, _W['out']),
_b['out']))
    out = {
        'input_r': _input_r, 'conv1': _conv1, 'pool1': _pool1, '
layer1': _layer1,
        'conv2': _conv2, 'pool2': _pool2, 'layer2': _layer2, 'de
nse': _dense,
        'out': _out
    }
    return out



# Define Parameter
learning_rate   = 0.001
training_epochs = 10
batch_size      = 100
display_step    = 1

# Define Functions
x = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.float32, [None, 10])
keepprob = tf.placeholder(tf.float32)
pred = mlp(x, weights, biases, keepprob)['out']
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pr
ed, y))
optm = tf.train.AdamOptimizer(learning_rate).minimize(cost)
corr = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accr = tf.reduce_mean(tf.cast(corr, tf.float32))
init = tf.initialize_all_variables()
```

```
print ("Net Ready")
```

```
Net Ready
```

```
# Let's do it!
sess = tf.Session()
sess.run(init)

# Training cycle
for epoch in range(training_epochs):
    sum_cost = 0.
    total_batch = int(mnist.train.num_examples/batch_size)
    # Loop over all batches
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        # Fit training using batch data
        sess.run(optm, feed_dict={x: batch_xs, y: batch_ys, keep
prob: 0.7})
        # Compute average loss
        curr_cost = sess.run(cost, feed_dict={x: batch_xs, y: ba
tch_ys, keepprob: 1.})
        sum_cost = sum_cost + curr_cost
    avg_cost = sum_cost / total_batch

    # Display logs per epoch step
    if epoch % display_step == 0:
        print ("Epoch: %03d/%03d cost: %.9f" % (epoch, training_
epochs, avg_cost))
        train_acc = sess.run(accr, feed_dict={x: batch_xs, y: ba
tch_ys, keepprob: 1.})
        print ("Training accuracy: %.3f" % (train_acc))
        test_acc = sess.run(accr, feed_dict={x: testimgs, y: tes
tlabels, keepprob: 1.})
        print ("Test accuracy: %.3f" % (test_acc))

print ("Optimization Finished!")
```

```
# Get Random Image
randidx = np.random.randint(ntest)
testimg = testimgs[randidx:randidx+1, :]

# Run Network
inputimg = sess.run(mlp(x, weights, biases, keepprob)['input_r']
,
                feed_dict={x: testimg, keepprob: 1.})
outval = sess.run(mlp(x, weights, biases, keepprob)['out'],
                feed_dict={x: testimg, keepprob: 1.})
camval = sess.run(mlp(x, weights, biases, keepprob)['conv2'],
                feed_dict={x: testimg, keepprob: 1.})
cweights = sess.run(weights['out'])

# Plot original Image
plt.matshow(inputimg[0, :, :, 0], cmap=plt.get_cmap('gray'))
plt.title("Input image")
plt.colorbar()
plt.show()

# Plot class activation maps
fig, axs = plt.subplots(2, 5, figsize=(15, 6))
for i in range(10):
    predlabel   = np.argmax(outval)
    predweights = cweights[:, i:i+1]
    camsum = np.zeros((14, 14))
    for j in range(256):
        camsum = camsum + predweights[j]*camval[0, :, :, j]
    camavg = camsum / 256
    # Plot
    im = axs[int(i/5)][i%5].matshow(camavg, cmap=plt.get_cmap('g
ray'))
    axs[int(i/5)][i%5].set_title(("[%d] prob is %.3f") % (i, out
val[0, i]))
    plt.draw()

fig.subplots_adjust(right=0.8)
cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
fig.colorbar(im, cax=cbar_ax)
```

TensorBoard Usage

Visualizing Linear Regression

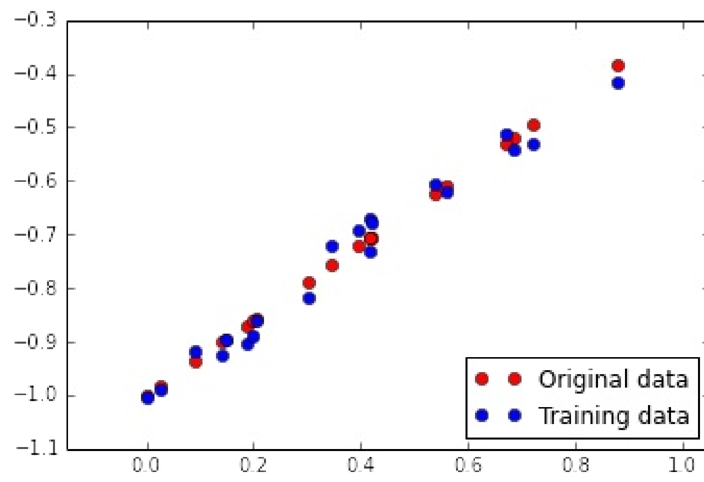
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

np.random.seed(1)
def f(x, a, b):
    n    = train_X.size
    vals = np.zeros((1, n))
    for i in range(0, n):
        ax  = np.multiply(a, x.item(i))
        val = np.add(ax, b)
        vals[0, i] = val
    return vals

Wref = 0.7
bref = -1.
n    = 20
noise_var = 0.001
train_X   = np.random.random((1, n))
ref_Y     = f(train_X, Wref, bref)
train_Y   = ref_Y + np.sqrt(noise_var)*np.random.randn(1, n)
n_samples = train_X.size

# Plot
plt.figure(1)
plt.plot(train_X[0, :], ref_Y[0, :], 'ro', label='Original data'
)
plt.plot(train_X[0, :], train_Y[0, :], 'bo', label='Training dat
a')
plt.axis('equal')
plt.legend(loc='lower right')
```

```
<matplotlib.legend.Legend at 0x7f77fa9eb3d0>
```



−0.3
−0.4
−0.5
−0.6
−0.7
−0.8
−0.9
−1.0
−1.1
0.0
0.2
0.4
0.6
0.8
1.0
Original data
Training data

```
# Parameters
training_epochs = 1000
display_step    = 100

# Set TensorFlow Graph
x = tf.placeholder(tf.float32, name="INPUT_x")
y = tf.placeholder(tf.float32, name="OUTPUT_y")
W = tf.Variable(np.random.randn(), name="WEIGHT_W")
b = tf.Variable(np.random.randn(), name="BIAS_b")

# Construct a Model
activation = tf.add(tf.mul(x, W), b)

# Define Error Measure and Optimizer
learning_rate   = 0.01
cost = tf.reduce_mean(tf.pow(activation-y, 2))
optimizer = tf.train.GradientDescentOptimizer(learning_rate).min
imize(cost) #Gradient descent

# Initializer
init = tf.initialize_all_variables()

# Run!
sess = tf.Session()
# Initialize
sess.run(init)
# Summary
summary_writer = tf.train.SummaryWriter('/tmp/tf_logs/linear_reg
ression', graph=sess.graph)
for epoch in range(training_epochs):
    for (_x, _y) in zip(train_X[0, :], train_Y[0, :]):
        # print "x: ", x, " y: ", y
        sess.run(optimizer, feed_dict={x:_x, y:_y})

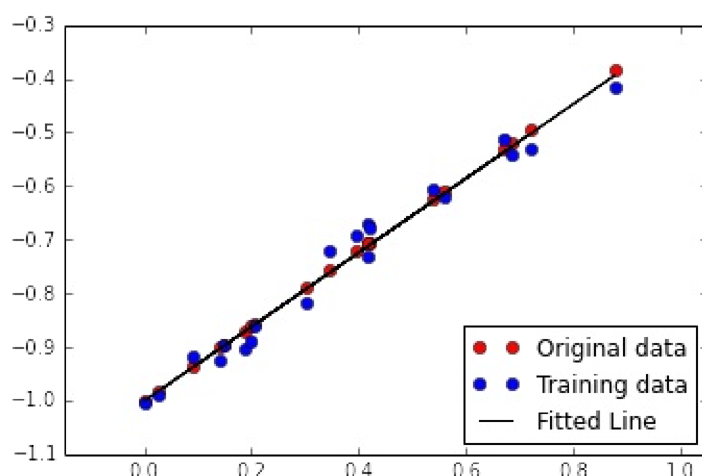
    # Check cost
    if epoch % display_step == 0:
        costval = sess.run(cost, feed_dict={x: train_X, y:train_
Y})
        print("[%d/%d] cost :%.3f" % (epoch, training_epochs, co
stval)),
        Wtemp = sess.run(W)
        btemp = sess.run(b)
        print("Wtemp is %.3f and Wref is %.3f" % (Wtemp, Wref)),
        print("btemp is %.3f and bref is %.3f" % (btemp, bref))

# Final W and b
Wopt = sess.run(W)
bopt = sess.run(b)
fopt = f(train_X, Wopt, bopt)
```

```
[0/1000] cost :0.215 Wtemp is -0.696 and Wref is 0.700 btemp is
-0.172 and bref is -1.000
[100/1000] cost :0.003 Wtemp is 0.505 and Wref is 0.700 btemp is
 -0.928 and bref is -1.000
[200/1000] cost :0.001 Wtemp is 0.667 and Wref is 0.700 btemp is
 -0.992 and bref is -1.000
[300/1000] cost :0.001 Wtemp is 0.688 and Wref is 0.700 btemp is
 -1.000 and bref is -1.000
[400/1000] cost :0.001 Wtemp is 0.690 and Wref is 0.700 btemp is
 -1.001 and bref is -1.000
[500/1000] cost :0.001 Wtemp is 0.690 and Wref is 0.700 btemp is
 -1.001 and bref is -1.000
[600/1000] cost :0.001 Wtemp is 0.691 and Wref is 0.700 btemp is
 -1.001 and bref is -1.000
[700/1000] cost :0.001 Wtemp is 0.691 and Wref is 0.700 btemp is
 -1.001 and bref is -1.000
[800/1000] cost :0.001 Wtemp is 0.691 and Wref is 0.700 btemp is
 -1.001 and bref is -1.000
[900/1000] cost :0.001 Wtemp is 0.691 and Wref is 0.700 btemp is
 -1.001 and bref is -1.000
```

```
# Plot Results
plt.figure(2)
plt.plot(train_X[0, :], ref_Y[0, :], 'ro', label='Original data'
)
plt.plot(train_X[0, :], train_Y[0, :], 'bo', label='Training dat
a')
plt.plot(train_X[0, :], fopt[0, :], 'k-', label='Fitted Line')
plt.axis('equal')
plt.legend(loc='lower right')
```

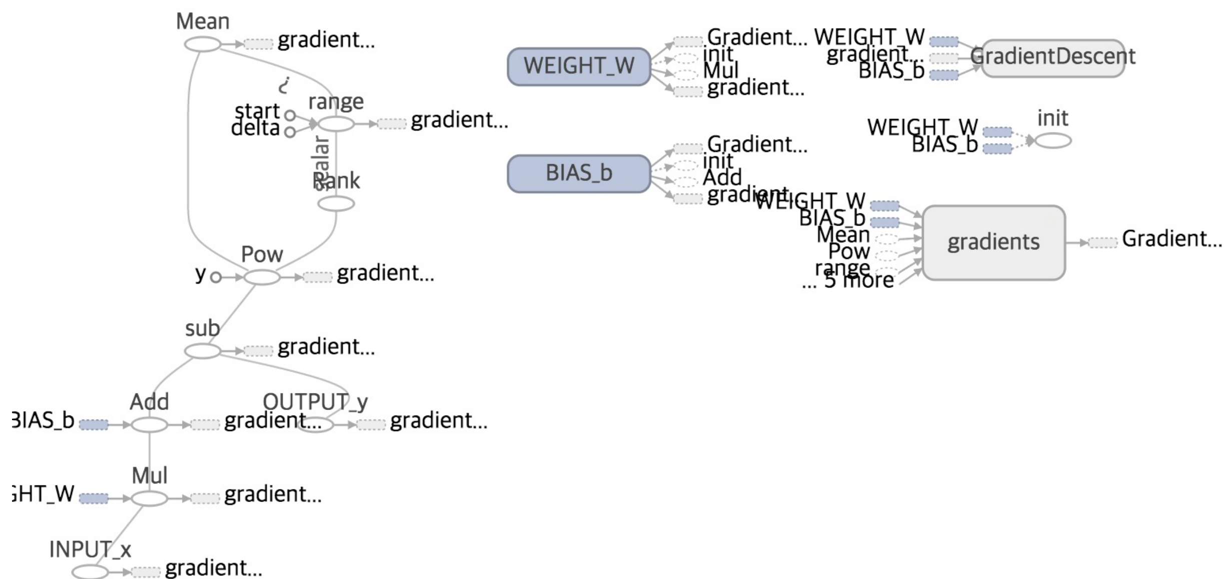
```
<matplotlib.legend.Legend at 0x7f77cc2d8990>
```



Run the command line

tensorboard --logdir=/tmp/tf_logs/linear_regression

Open http://localhost:6006/ into your web browser



Visualizing MNIST + Summary

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
%matplotlib inline
mnist = input_data.read_data_sets('data/', one_hot=True)
trainimg   = mnist.train.images
trainlabel = mnist.train.labels
testimg    = mnist.test.images
testlabel  = mnist.test.labels
print ("Packages loaded")
```

```
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
Packages loaded
```

Define networks

tf.variable_scope("NAME"):

```
# Parameters
learning_rate   = 0.001
training_epochs = 10
batch_size      = 100
display_step    = 2
# Network Parameters
n_input    = 784 # MNIST data input (img shape: 28*28)
n_hidden_1 = 256 # 1st layer num features
n_hidden_2 = 256 # 2nd layer num features
n_hidden_3 = 256 # 3rd layer num features
n_hidden_4 = 256 # 4th layer num features
n_classes  = 10 # MNIST total classes (0-9 digits)

# Store layers weight & bias
stddev = 0.1
with tf.variable_scope("MLP_WEIGHTS"):
    weights = {
        'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1]
, stddev=stddev)),
        'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden
_2], stddev=stddev)),
        'h3': tf.Variable(tf.random_normal([n_hidden_2, n_hidden
_3], stddev=stddev)),
        'h4': tf.Variable(tf.random_normal([n_hidden_3, n_hidden
_4], stddev=stddev)),
        'out': tf.Variable(tf.random_normal([n_hidden_4, n_class
es], stddev=stddev))
    }
with tf.variable_scope("MLP_BIASES"):
    biases = {
        'b1': tf.Variable(tf.random_normal([n_hidden_1])),
        'b2': tf.Variable(tf.random_normal([n_hidden_2])),
        'b3': tf.Variable(tf.random_normal([n_hidden_3])),
        'b4': tf.Variable(tf.random_normal([n_hidden_4])),
        'out': tf.Variable(tf.random_normal([n_classes]))
    }
```

```
# Create model
def multilayer_perceptron(_X, _weights, _biases, _keep_prob):
    with tf.variable_scope("MLP_LAYER_1"):
        layer_1 = tf.nn.dropout(tf.nn.relu(tf.add(tf.matmul(_X,
_weights['h1'])
                                                  , _biases['b1'
])), _keep_prob)
    with tf.variable_scope("MLP_LAYER_2"):
        layer_2 = tf.nn.dropout(tf.nn.relu(tf.add(tf.matmul(laye
r_1, _weights['h2'])
                                                  , _biases['b2'
])), _keep_prob)
    with tf.variable_scope("MLP_LAYER_3"):
        layer_3 = tf.nn.dropout(tf.nn.relu(tf.add(tf.matmul(laye
r_2, _weights['h3'])
                                                  , _biases['b3'
])), _keep_prob)
    with tf.variable_scope("MLP_LAYER_OUT"):
        layer_4 = tf.nn.dropout(tf.nn.relu(tf.add(tf.matmul(laye
r_3, _weights['h4'])
                                                  , _biases['b4'
])), _keep_prob)
    return (tf.matmul(layer_4, _weights['out']) + _biases['out']
)
print ("Network ready")
```

```
Network ready
```

Define functions

```
# tf Graph input
x = tf.placeholder("float", [None, n_input], name="MLP_INPUT_x")
y = tf.placeholder("float", [None, n_classes], name="MLP_TARGET_
y")
keepratio = tf.placeholder("float", name="MLP_DROPOUT")
# Construct model
pred = multilayer_perceptron(x, weights, biases, keepratio)
# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pr
ed, y)) # Softmax loss
optm = tf.train.AdamOptimizer(learning_rate=learning_rate).minim
ize(cost) # Adam Optimizer
# Accuracy
corr = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accr = tf.reduce_mean(tf.cast(corr, "float"))
# Initializing the variables
init = tf.initialize_all_variables()
print ("Network Ready")
```

```
Network Ready
```

Summary

```
# Do some optimizations
sess = tf.Session()
sess.run(init)

# Summary writer
tf.scalar_summary('cross entropy', cost)
tf.scalar_summary('accuracy', accr)
merged = tf.merge_all_summaries()
summary_writer = tf.train.SummaryWriter('/tmp/tf_logs/mlp_mnist'
                                        , graph=sess.graph)
print ("Summary ready")
```

```
Summary ready
```

Run

```
print ("Start!")
# Training cycle
for epoch in range(training_epochs):
    avg_cost = 0.
    total_batch = int(mnist.train.num_examples/batch_size)
    # Loop over all batches
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        # Fit training using batch data
        summary, _ = sess.run([merged, optm]
                , feed_dict={x: batch_xs, y: batch_ys, keepratio
: 0.7})
        # Compute average loss
        avg_cost += sess.run(cost
                , feed_dict={x: batch_xs, y: batch_ys, keepratio:
1.})/total_batch
        # Add summary
        summary_writer.add_summary(summary, epoch*total_batch+i)
    # Display logs per epoch step
    if epoch % display_step == 0:
        print ("Epoch: %03d/%03d cost: %.9f" % (epoch, training_
epochs, avg_cost))
        train_acc = sess.run(accr, feed_dict={x: batch_xs, y: ba
tch_ys, keepratio:1.})
        print (" Training accuracy: %.3f" % (train_acc))
        test_acc = sess.run(accr, feed_dict={x: testimg, y: test
label, keepratio:1.})
        print (" Test accuracy: %.3f" % (test_acc))

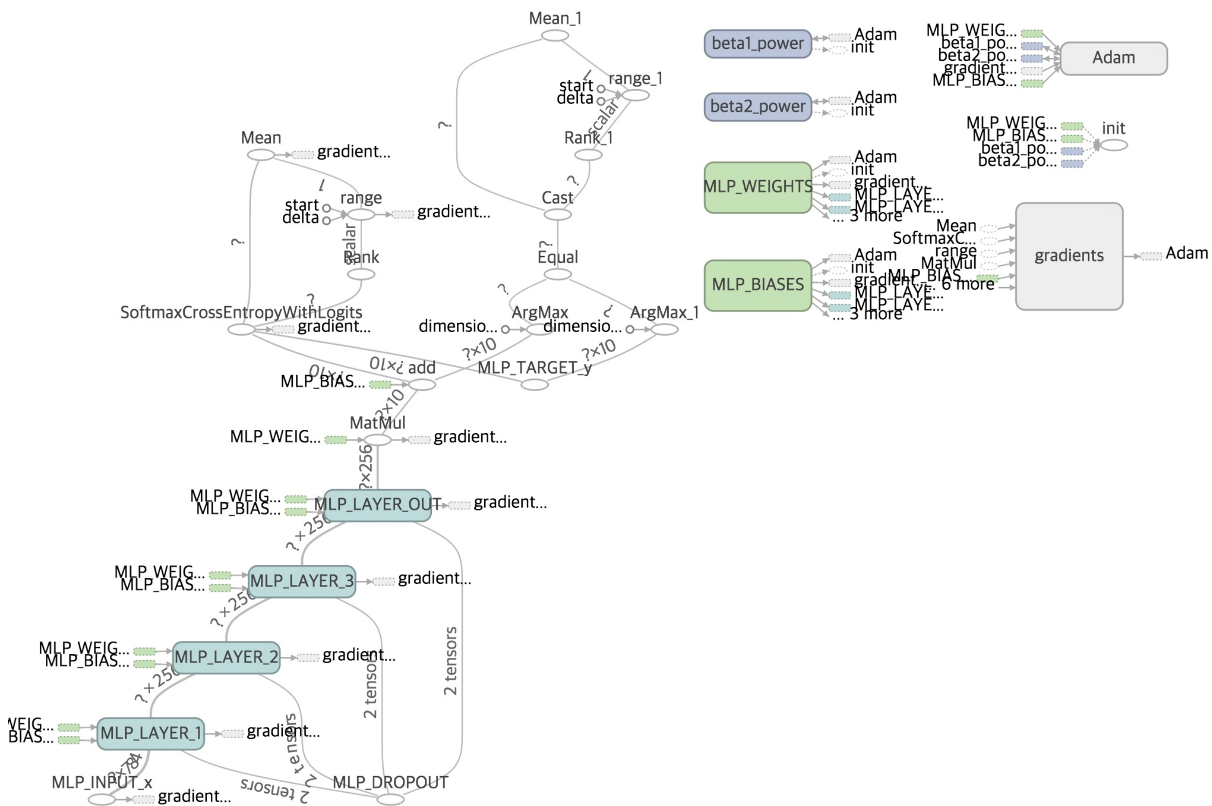
print ("Optimization Finished!")
```

```
Start!
Epoch: 000/010 cost: 0.475366192
 Training accuracy: 0.950
 Test accuracy: 0.929
Epoch: 002/010 cost: 0.118928117
 Training accuracy: 0.970
 Test accuracy: 0.961
Epoch: 004/010 cost: 0.073962761
 Training accuracy: 1.000
 Test accuracy: 0.973
Epoch: 006/010 cost: 0.051166516
 Training accuracy: 0.980
 Test accuracy: 0.976
Epoch: 008/010 cost: 0.038799497
 Training accuracy: 0.990
 Test accuracy: 0.980
Optimization Finished!
```

Run the command line

tensorboard --logdir=/tmp/tf_logs/mlp_mnist

Open http://localhost:6006/ into your web browser



Visualizing CNN + Summary

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
%matplotlib inline
mnist = input_data.read_data_sets('data/', one_hot=True)
trainimg   = mnist.train.images
trainlabel = mnist.train.labels
testimg    = mnist.test.images
testlabel  = mnist.test.labels
print ("Packages loaded.")
```

```
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
Packages loaded.
```

Define Networks

tf.variable_scope("NAME"):

```
# Parameters
learning_rate   = 0.001
training_epochs = 5
batch_size      = 100
display_step    = 1

# Network
n_input  = 784
n_output = 10
with tf.variable_scope("CNN_WEIGHTS"):
    weights  = {
        'wc1': tf.Variable(tf.random_normal([3, 3, 1, 64], stdde
v=0.1)),
        'wc2': tf.Variable(tf.random_normal([3, 3, 64, 128], std
dev=0.1)),
        'wd1': tf.Variable(tf.random_normal([7*7*128, 1024], std
dev=0.1)),
        'wd2': tf.Variable(tf.random_normal([1024, n_output], st
ddev=0.1))
    }
with tf.variable_scope("CNN_BIASES"):
    biases   = {
        'bc1': tf.Variable(tf.random_normal([64], stddev=0.1)),
        'bc2': tf.Variable(tf.random_normal([128], stddev=0.1)),
        'bd1': tf.Variable(tf.random_normal([1024], stddev=0.1))
,
        'bd2': tf.Variable(tf.random_normal([n_output], stddev=0
.1))
    }
```

```
def conv_basic(_input, _w, _b, _keepratio):
    # Input
    with tf.variable_scope("INPUT_LAYER"):
        _input_r = tf.reshape(_input, shape=[-1, 28, 28, 1])
    # Conv1
    with tf.variable_scope("CNN_CONV_1"):
        _conv1 = tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(_input_r
, _w['wc1']
                    , strides=[1, 1, 1, 1], padding='SAME'), _b[
'bc1']))
    with tf.variable_scope("CNN_POOL_1"):
        _pool1 = tf.nn.max_pool(_conv1, ksize=[1, 2, 2, 1], stri
des=[1, 2, 2, 1]
                                , padding='SAME')
        _pool_dr1 = tf.nn.dropout(_pool1, _keepratio)
    # Conv2
    with tf.variable_scope("CNN_CONV_2"):
        _conv2 = tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(_pool_dr
1, _w['wc2']
                    , strides=[1, 1, 1, 1], padding='SAME'), _b[
'bc2']))
    with tf.variable_scope("CNN_POOL_2"):
        _pool2 = tf.nn.max_pool(_conv2, ksize=[1, 2, 2, 1], stri
des=[1, 2, 2, 1]
                                , padding='SAME')
        _pool_dr2 = tf.nn.dropout(_pool2, _keepratio)
    with tf.variable_scope("FC_1"):
        # Vectorize
        _dense1 = tf.reshape(_pool_dr2, [-1, _w['wd1'].get_shape
().as_list()[0]])
        # Fc1
        _fc1 = tf.nn.relu(tf.nn.bias_add(tf.matmul(_dense1, _w['
wd1']), _b['bd1']))
        _fc_dr1 = tf.nn.dropout(_fc1, _keepratio)
    with tf.variable_scope("FC_2"):
        # Fc2
        _out = tf.add(tf.matmul(_fc_dr1, _w['wd2']), _b['bd2'])
    # Return everything
    out = {
        'input_r': _input_r, 'conv1': _conv1, 'pool1': _pool1, '
pool1_dr1': _pool_dr1,
        'conv2': _conv2, 'pool2': _pool2, 'pool_dr2': _pool_dr2,
'dense1': _dense1,
        'fc1': _fc1, 'fc_dr1': _fc_dr1, 'out': _out }
    return out

print ("Network ready")
```

```
Network ready
```

Define functions

```
# tf Graph input
x = tf.placeholder(tf.float32, [None, n_input], name="CNN_INPUT_
x")
y = tf.placeholder(tf.float32, [None, n_output], name="CNN_TARGE
T_y")
keepratio = tf.placeholder(tf.float32, name="CNN_DROPOUT_keeprat
io")

# Functions!
pred = conv_basic(x, weights, biases, keepratio)['out']
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pr
ed, y))
optm = tf.train.AdamOptimizer(learning_rate=learning_rate).minim
ize(cost)
corr = tf.equal(tf.argmax(pred,1), tf.argmax(y,1)) # Count corre
cts
accr = tf.reduce_mean(tf.cast(corr, tf.float32)) # Accuracy
init = tf.initialize_all_variables()
print ("Functions ready")
```

```
Functions ready
```

Summary

```
# Do some optimizations
sess = tf.Session()
sess.run(init)

# Summary writer
tf.scalar_summary('cross entropy', cost)
tf.scalar_summary('accuracy', accr)
merged = tf.merge_all_summaries()
summary_writer = tf.train.SummaryWriter('/tmp/tf_logs/cnn_mnist'
                                        , graph=sess.graph)
print ("Summary ready")
```

```
Summary ready
```

Run

```
print ("Start!")
for epoch in range(training_epochs):
    avg_cost = 0.
    total_batch = int(mnist.train.num_examples/batch_size)
    # Loop over all batches
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        # Fit training using batch data
        summary, _ = sess.run([merged, optm]
                               , feed_dict={x: batch_xs, y: batch_ys, k
eepratio:0.7})
        # Compute average loss
        avg_cost += sess.run(cost
                     , feed_dict={x: batch_xs, y: batch_ys, keepratio:
1.})/total_batch
        # Add summary
        summary_writer.add_summary(summary, epoch*total_batch+i)
    # Display logs per epoch step
    if epoch % display_step == 0:
        print ("Epoch: %03d/%03d cost: %.9f" % (epoch, training_
epochs, avg_cost))
        train_acc = sess.run(accr, feed_dict={x: batch_xs, y: ba
tch_ys, keepratio:1.})
        print (" Training accuracy: %.3f" % (train_acc))
        test_acc = sess.run(accr, feed_dict={x: testimg, y: test
label, keepratio:1.})
        print (" Test accuracy: %.3f" % (test_acc))

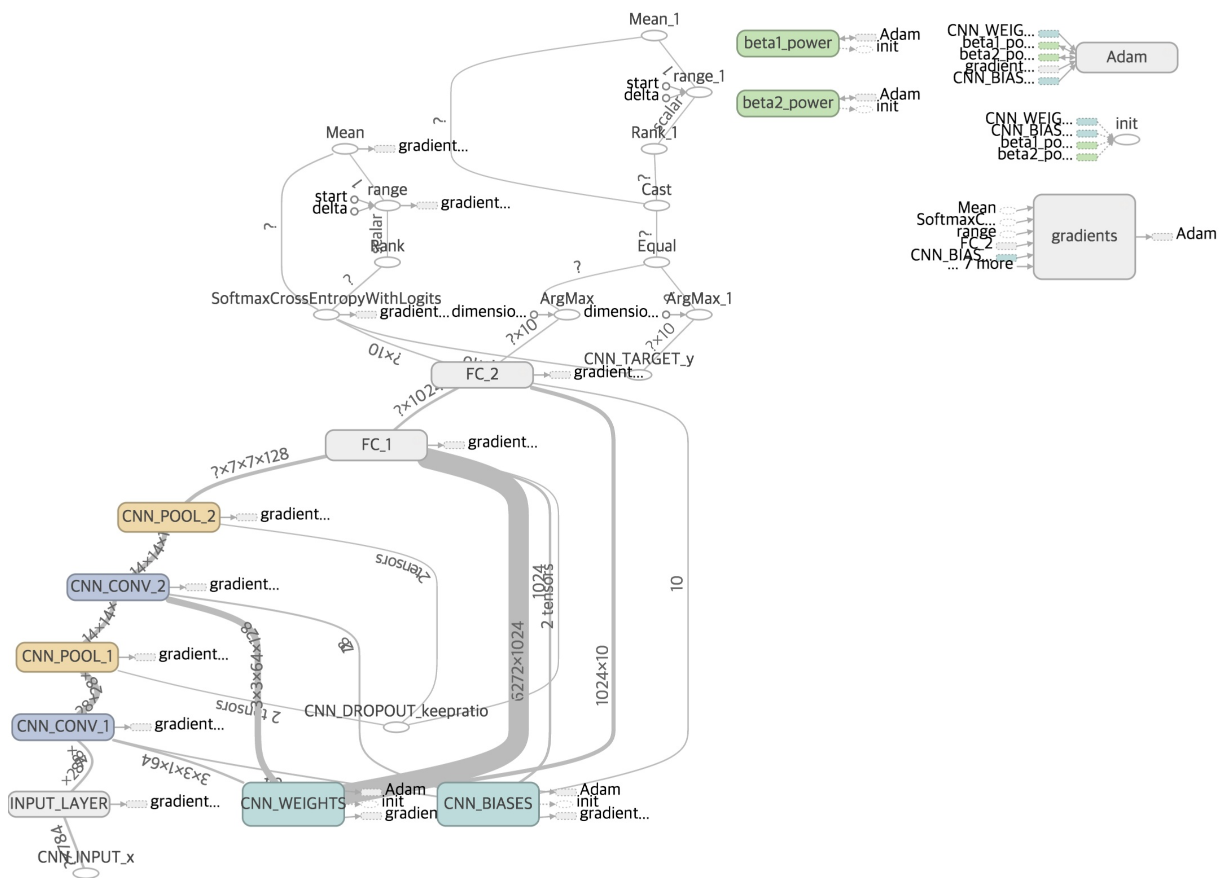
print ("Optimization Finished.")
```

```
Start!
Epoch: 000/005 cost: 0.352900247
 Training accuracy: 0.990
 Test accuracy: 0.981
Epoch: 001/005 cost: 0.047190106
 Training accuracy: 0.980
 Test accuracy: 0.988
Epoch: 002/005 cost: 0.031958400
 Training accuracy: 1.000
 Test accuracy: 0.990
Epoch: 003/005 cost: 0.022927662
 Training accuracy: 1.000
 Test accuracy: 0.990
Epoch: 004/005 cost: 0.018374201
 Training accuracy: 0.990
 Test accuracy: 0.991
Optimization Finished.
```

Run the command line

tensorboard --logdir=/tmp/tf_logs/cnn_mnist

Open http://localhost:6006/ into your web browser



Basic semantic segmentation using average unpooling

```
from PIL import Image
import cPickle as pkl
import time
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.python.training import moving_averages
import tensorflow as tf
import glob
import os
%matplotlib inline
print ("Packs loaded.")
```

```
Packs loaded.
```

Load dataset for semantic segmentation

```
# Location of the files
camvidpath = 'data/seg/SegNet-Tutorial-master/CamVid/'
# Training data
path1 = os.getcwd() + '/' + camvidpath + 'train/'
path2 = os.getcwd() + '/' + camvidpath + 'trainannot/'
trainimglist = glob.glob(path1 + '/*.png')
trainannotlist = glob.glob(path2 + '/*.png')
print ("%d train images" % (len(trainimglist)))
print ("%d train annotations" % (len(trainannotlist)))

# Test data
path1 = os.getcwd() + '/' + camvidpath + 'test/'
path2 = os.getcwd() + '/' + camvidpath + 'testannot/'
testimglist = glob.glob(path1 + '/*.png')
testannotlist = glob.glob(path2 + '/*.png')
print ("%d test images" % (len(testimglist)))
print ("%d test annotations" % (len(testannotlist)))
```

```
367 train images
367 train annotations
233 test images
233 test annotations
```

Get train / test images

```
height = 128
width = 128
nrclass = 22
trainData = None
trainLabel = None
trainLabelOneHot = None
trainlen = len(trainimglist)
testData = None
testLabel = None
testLabelOneHot = None
testlen = len(testimglist)
def DenseToOneHot(labels_dense, num_classes):
    # Convert class labels from scalars to one-hot vectors.
    num_labels = labels_dense.shape[0]
    index_offset = np.arange(num_labels) * num_classes
    labels_one_hot = np.zeros((num_labels, num_classes))
    labels_one_hot.flat[index_offset + labels_dense.ravel()] = 1
    return labels_one_hot
""" Train data process """
for (f1, f2, i) in zip(trainimglist, trainannotlist, range(train
len)):
    # print ("[%02d/%02d]f1: %sf2: %s" % (i, trainlen, f1, f2))
    # Train image
    img1 = Image.open(f1)
    img1 = img1.resize((height, width))
    rgb  = np.array(img1).reshape(1, height, width, 3)
    # Train label
    img2 = Image.open(f2)
    img2 = img2.resize((height, width), Image.NEAREST)
    label = np.array(img2).reshape(1, height, width, 1)
    # Stack images and labels
    if i == 0:
        trainData = rgb
        trainLabel = label
    else:
        trainData = np.concatenate((trainData, rgb), axis=0)
        trainLabel = np.concatenate((trainLabel, label), axis=0)
ntrain = len(trainData)
# Onehot-coded label
trainLabelOneHot = np.zeros((trainLabel.shape[0], trainLabel.sha
pe[1]                             , trainLabel.shape[2], nrclass
))
for row in range(height):
    for col in range(width):
        single = trainLabel[:, row, col, 0]
        oneHot = DenseToOneHot(single, nrclass) # (367,) => (367
, 22)
        trainLabelOneHot[:, row, col, :] = oneHot
print ("Train data process done.")
```

```
""" Test data process """
for (f1, f2, i) in zip(testimglist, testannotlist, range(testlen
)):
    # print ("[%02d/%02d]f1: %sf2: %s" % (i, testlen, f1, f2))
    # Train image
    img1 = Image.open(f1)
    img1 = img1.resize((height, width))
    rgb  = np.array(img1).reshape(1, height, width, 3)
    # Train label
    img2 = Image.open(f2)
    img2 = img2.resize((height, width), Image.NEAREST)
    label = np.array(img2).reshape(1, height, width, 1)
    # Stack images and labels
    if i == 0:
        testData = rgb
        testLabel = label
    else:
        testData = np.concatenate((testData, rgb), axis=0)
        testLabel = np.concatenate((testLabel, label), axis=0)
# Onehot-coded label
testLabelOneHot = np.zeros((testLabel.shape[0], testLabel.shape[1
], testLabel.shape[2], nrclass))
for row in range(height):
    for col in range(width):
        single = testLabel[:, row, col, 0]
        oneHot = DenseToOneHot(single, nrclass) # (367,) => (367
, 22)
        testLabelOneHot[:, row, col, :] = oneHot
print ("Test data process done.")
```

```
Train data process done.
Test data process done.
```

```
print ("Shape of 'trainData' is %s" % (trainData.shape,))
print ("Shape of 'trainLabel' is %s" % (trainLabel.shape,))
print ("Shape of 'trainLabelOneHot' is %s" % (trainLabelOneHot.s
hape,))
print ("Shape of 'testData' is %s" % (testData.shape,))
print ("Shape of 'testLabel' is %s" % (testLabel.shape,))
print ("Shape of 'testLabelOneHot' is %s" % (testLabelOneHot.sha
pe,))
```

```
Shape of 'trainData' is (367, 128, 128, 3)
Shape of 'trainLabel' is (367, 128, 128, 1)
Shape of 'trainLabelOneHot' is (367, 128, 128, 22)
Shape of 'testData' is (233, 128, 128, 3)
Shape of 'testLabel' is (233, 128, 128, 1)
Shape of 'testLabelOneHot' is (233, 128, 128, 22)
```

Define networks

```
# Define functions
x = tf.placeholder(tf.float32, [None, height, width, 3])
y = tf.placeholder(tf.float32, [None, height, width, nrclass])
keepprob = tf.placeholder(tf.float32)
# Kernels
ksize = 5
fsize = 64
initstdev = 0.01
initfun = tf.random_normal_initializer(mean=0.0, stddev=initstde
v)
# initfun = None
weights = {
    'ce1': tf.get_variable("ce1", shape = [ksize, ksize, 3, fsiz
e], initializer = initfun) ,
    'ce2': tf.get_variable("ce2", shape = [ksize, ksize, fsize,
fsize], initializer = initfun) ,
    'ce3': tf.get_variable("ce3", shape = [ksize, ksize, fsize,
fsize], initializer = initfun),
    'ce4': tf.get_variable("ce4", shape = [ksize, ksize, fsize,
fsize], initializer = initfun),
    'cd4': tf.get_variable("cd4", shape = [ksize, ksize, fsize,
fsize], initializer = initfun),
    'cd3': tf.get_variable("cd3", shape = [ksize, ksize, fsize,
fsize], initializer = initfun),
    'cd2': tf.get_variable("cd2", shape = [ksize, ksize, fsize,
fsize], initializer = initfun),
    'cd1': tf.get_variable("cd1", shape = [ksize, ksize, fsize,
fsize], initializer = initfun),
    'dense_inner_prod': tf.get_variable("dense_inner_prod", shap
e= [1, 1, fsize, nrclass]
                                       , initializer = initfun)
# <= 1x1conv
}
biases = {
    'be1': tf.get_variable("be1", shape = [fsize], initializer =
 tf.constant_initializer(value=0.0)),
    'be2': tf.get_variable("be2", shape = [fsize], initializer =
 tf.constant_initializer(value=0.0)),
    'be3': tf.get_variable("be3", shape = [fsize], initializer =
 tf.constant_initializer(value=0.0)),
    'be4': tf.get_variable("be4", shape = [fsize], initializer =
 tf.constant_initializer(value=0.0)),
    'bd4': tf.get_variable("bd4", shape = [fsize], initializer =
 tf.constant_initializer(value=0.0)),
    'bd3': tf.get_variable("bd3", shape = [fsize], initializer =
 tf.constant_initializer(value=0.0)),
    'bd2': tf.get_variable("bd2", shape = [fsize], initializer =
 tf.constant_initializer(value=0.0)),
    'bd1': tf.get_variable("bd1", shape = [fsize], initializer =
 tf.constant_initializer(value=0.0))
}
```

DeconvNet model

```
# input : [m, h, w, c]
def Unpooling(inputOrg, size, mask=None):
    # m, c, h, w order
    m = size[0]
    h = size[1]
    w = size[2]
    c = size[3]
    input = tf.transpose(inputOrg, [0, 3, 1, 2])
    x = tf.reshape(input, [-1, 1])
    k = np.float32(np.array([1.0, 1.0]).reshape([1,-1]))
    output = tf.matmul(x, k)
    output = tf.reshape(output,[-1, c, h, w * 2])
    # m, c, w, h
    xx = tf.transpose(output, [0, 1, 3, 2])
    xx = tf.reshape(xx,[-1, 1])
    output = tf.matmul(xx, k)
    # m, c, w, h
    output = tf.reshape(output, [-1, c, w * 2, h * 2])
    output = tf.transpose(output, [0, 3, 2, 1])
    outshape = tf.pack([m, h * 2, w * 2, c])
    if mask != None:
        dense_mask = tf.sparse_to_dense(mask, outshape, output, 0
)
        return output, dense_mask
    else:
        return output

# DeconvNet Model
def Model(_X, _W, _b, _keepprob):
    use_bias = 1
    # Encoder 128x128
    encoder1 = tf.nn.conv2d(_X, _W['ce1'], strides=[1, 1, 1, 1],
 padding='SAME')
    if use_bias:
        encoder1 = tf.nn.bias_add(encoder1, _b['be1'])
    mean, var = tf.nn.moments(encoder1, [0, 1, 2])
    encoder1 = tf.nn.batch_normalization(encoder1, mean, var, 0,
1, 0.0001)
    encoder1 = tf.nn.relu(encoder1)
    encoder1 = tf.nn.max_pool(encoder1, ksize=[1, 2, 2, 1], stri
des=[1, 2, 2, 1], padding='SAME')
    encoder1 = tf.nn.dropout(encoder1, _keepprob)
    # 64x64
    encoder2 = tf.nn.conv2d(encoder1, _W['ce2'], strides=[1, 1, 1
, 1], padding='SAME')
    if use_bias:
        encoder2 = tf.nn.bias_add(encoder2, _b['be2'])
    mean, var = tf.nn.moments(encoder1, [0, 1, 2])
    encoder2 = tf.nn.batch_normalization(encoder2, mean, var, 0,
```

```
1, 0.0001)
    encoder2 = tf.nn.relu(encoder2)
    encoder2 = tf.nn.max_pool(encoder2, ksize=[1, 2, 2, 1], stri
des=[1, 2, 2, 1], padding='SAME')
    encoder2 = tf.nn.dropout(encoder2, _keepprob)
    # 32x32
    encoder3 = tf.nn.conv2d(encoder2, _W['ce3'], strides=[1, 1, 1
, 1], padding='SAME')
    if use_bias:
        encoder3 = tf.nn.bias_add(encoder3, _b['be3'])
    mean, var = tf.nn.moments(encoder3, [0, 1, 2])
    encoder3 = tf.nn.batch_normalization(encoder3, mean, var, 0,
1, 0.0001)
    encoder3 = tf.nn.relu(encoder3)
    encoder3 = tf.nn.max_pool(encoder3, ksize=[1, 2, 2, 1], stri
des=[1, 2, 2, 1], padding='SAME')
    encoder3 = tf.nn.dropout(encoder3, _keepprob)
    # 16x16
    encoder4 = tf.nn.conv2d(encoder3, _W['ce4'], strides=[1, 1, 1
, 1], padding='SAME')
    if use_bias:
        encoder4 = tf.nn.bias_add(encoder4, _b['be4'])
    mean, var = tf.nn.moments(encoder4, [0, 1, 2])
    encoder4 = tf.nn.batch_normalization(encoder4, mean, var, 0,
1, 0.0001)
    encoder4 = tf.nn.relu(encoder4)
    encoder4 = tf.nn.max_pool(encoder4, ksize=[1, 2, 2, 1], stri
des=[1, 2, 2, 1], padding='SAME')
    encoder4 = tf.nn.dropout(encoder4, _keepprob)
    # 8x8

    # Decoder 8x8 (128/16 = 8) fsize: 64
    decoder4 = Unpooling(encoder4, [tf.shape(_X)[0], height / 16
, width / 16, fsize])
    decoder4 = tf.nn.conv2d_transpose(decoder4, _W['cd4']
                , tf.pack([tf.shape(_X)[0], ksize, ksize, fsize]
)
                , strides=[1, 1, 1, 1], padding='SAME')
    if use_bias:
        decoder4 = tf.nn.bias_add(decoder4, _b['bd4'])
    mean, var = tf.nn.moments(decoder4, [0, 1, 2])
    decoder4 = tf.nn.batch_normalization(decoder4, mean, var, 0,
1, 0.0001)
    decoder4 = tf.nn.relu(decoder4)
    decoder4 = tf.nn.dropout(decoder4, _keepprob)
    # 16x16
    decoder3 = Unpooling(encoder3, [tf.shape(_X)[0], height/8, w
idth/8, fsize])
    decoder3 = tf.nn.conv2d(decoder3, _W['cd3'], strides=[1, 1, 1
, 1], padding='SAME')

    if use_bias:
        decoder3 = tf.nn.bias_add(decoder3, _b['bd3'])
```

```
    mean, var = tf.nn.moments(decoder3, [0, 1, 2])
    decoder3 = tf.nn.batch_normalization(decoder3, mean, var, 0,
1, 0.0001)
    decoder3 = tf.nn.relu(decoder3)
    decoder3 = tf.nn.dropout(decoder3, _keepprob)
    # 32x32
    decoder2 = Unpooling(decoder3, [tf.shape(_X)[0], height/4, w
idth/4, fsize])
    decoder2 = tf.nn.conv2d(decoder2, _W['cd2'], strides=[1, 1, 1
, 1], padding='SAME')
    if use_bias:
        decoder2 = tf.nn.bias_add(decoder2, _b['bd2'])
    mean, var = tf.nn.moments(decoder2, [0, 1, 2])
    decoder2 = tf.nn.batch_normalization(decoder2, mean, var, 0,
1, 0.0001)
    decoder2 = tf.nn.relu(decoder2)
    decoder2 = tf.nn.dropout(decoder2, _keepprob)
    # 64x64
    decoder1 = Unpooling(decoder2, [tf.shape(_X)[0], height / 2,
 width / 2, fsize])
    decoder1 = tf.nn.conv2d(decoder1, _W['cd1'], strides=[1, 1, 1
, 1], padding='SAME')
    if use_bias:
        decoder1 = tf.nn.bias_add(decoder1, _b['bd1'])
    mean, var = tf.nn.moments(decoder1, [0, 1, 2])
    decoder1 = tf.nn.batch_normalization(decoder1, mean, var, 0,
1, 0.0001)
    decoder1 = tf.nn.relu(decoder1)
    decoder1 = tf.nn.dropout(decoder1, _keepprob)
    # 128x128
    output = tf.nn.conv2d(decoder1, _W['dense_inner_prod'], stri
des=[1, 1, 1, 1], padding='SAME')
    return output

print ("Network ready")
```

```
Network ready
```

```
ksize
```

```
5
```

Define functions

```
pred = Model(x, weights, biases, keepprob)
lin_pred = tf.reshape(pred, shape=[-1, nrclass])
lin_y = tf.reshape(y, shape=[-1, nrclass])
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(li
n_pred, lin_y))
# Class label
predmax = tf.argmax(pred, 3)
ymax = tf.argmax(y, 3)
# Accuracy
corr = tf.equal(tf.argmax(y,3), tf.argmax(pred, 3))
accr = tf.reduce_mean(tf.cast(corr, "float"))
# Optimizer
optm = tf.train.AdamOptimizer(0.0001).minimize(cost)
batch_size = 128
n_epochs = 1000

print ("Functions ready")
```

```
Functions ready
```

Real optimization starts here

```
resumeTraining = True
with tf.Session() as sess:
    # you need to initialize all variables
    tf.initialize_all_variables().run()
    saver = tf.train.Saver()
    checkpoint = tf.train.latest_checkpoint("nets/semseg_basic")
    print ("checkpoint: %s" % (checkpoint))
    if resumeTraining == False:
        print "Start from scratch"
    elif  checkpoint:
        print "Restoring from checkpoint", checkpoint
        saver.restore(sess, checkpoint)
    else:
        print "Couldn't find checkpoint to restore from. Startin
g over."

    for epoch_i in range(n_epochs):
        trainLoss = []; trainAcc = []
        num_batch = int(ntrain/batch_size)+1
        for _ in range(num_batch):
            randidx = np.random.randint(ntrain, size=batch_size)
            batchData = trainData[randidx]
            batchLabel = trainLabelOneHot[randidx]
            sess.run(optm, feed_dict={x: batchData, y: batchLabe
l, keepprob: 0.7}) # <== Optm is done here!
```

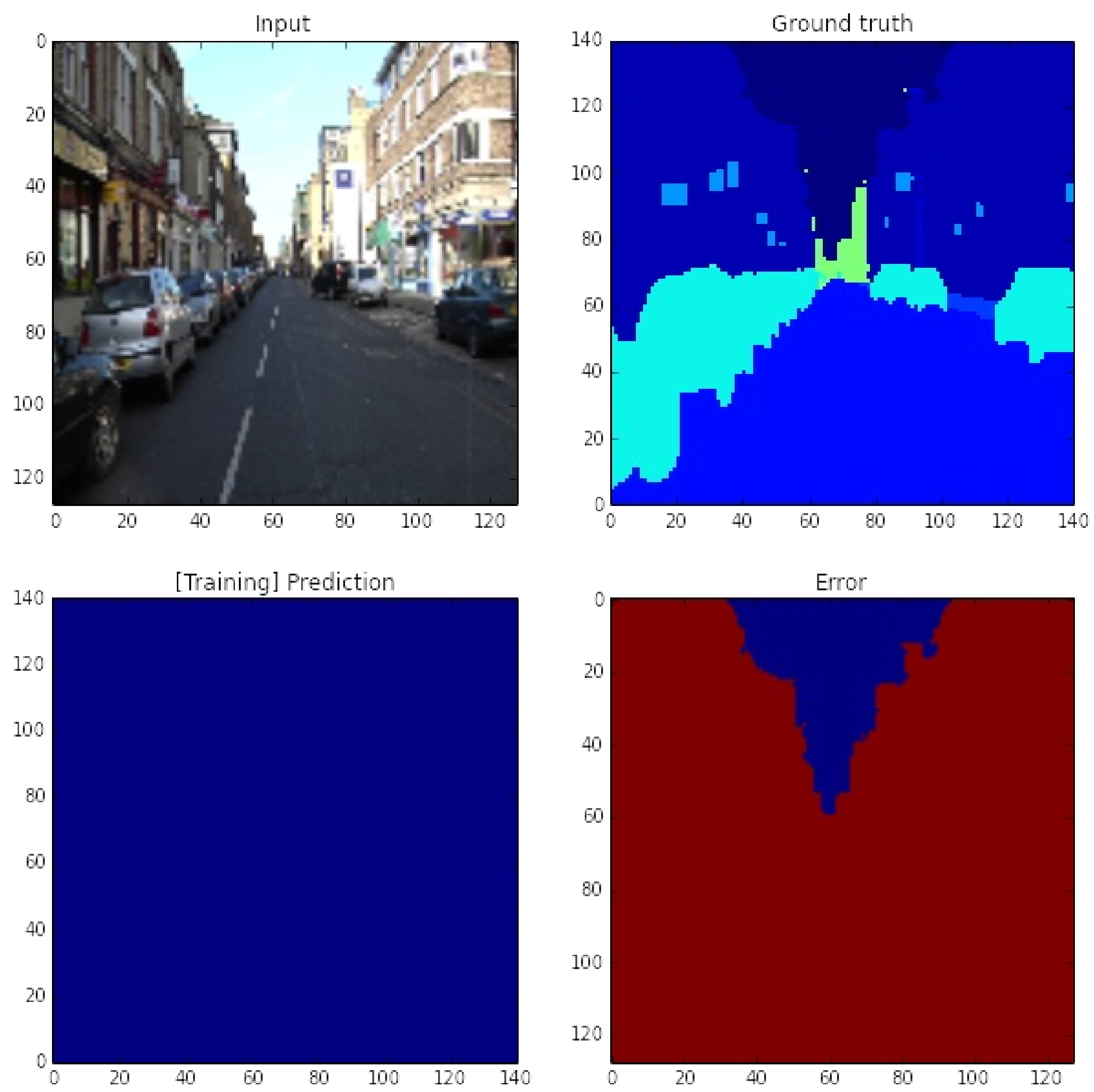
```
            trainLoss.append(sess.run(cost, feed_dict={x: batchD
ata, y: batchLabel, keepprob: 1.}))
            trainAcc.append(sess.run(accr, feed_dict={x: batchDa
ta, y: batchLabel, keepprob: 1.}))
        # Average loss and accuracy
        trainLoss = np.mean(trainLoss)
        trainAcc = np.mean(trainAcc)
        # Run test
        valLoss = sess.run(cost, feed_dict={x: testData, y: test
LabelOneHot, keepprob: 1.})
        valAcc = sess.run(accr, feed_dict={x: testData, y: testL
abelOneHot, keepprob: 1.})
        print ("[%02d/%02d] trainLoss: %.4f trainAcc: %.2f valLo
ss: %.4f valAcc: %.2f"
               % (epoch_i, n_epochs, trainLoss, trainAcc, valLos
s, valAcc))
        # Save snapshot
        if resumeTraining and epoch_i % 10 == 0:
            # Save
            saver.save(sess, 'nets/semseg_basic/progress', globa
l_step = epoch_i)
            # Train data
            index = np.random.randint(trainData.shape[0])
            refimg = trainData[index, :, :, :].reshape(height, w
idth, 3)
            batchData = trainData[index:index+1]
            batchLabel = trainLabelOneHot[index:index+1]
            predMaxOut = sess.run(predmax, feed_dict={x: batchDa
ta, y: batchLabel, keepprob:1.})
            yMaxOut = sess.run(ymax, feed_dict={x: batchData, y:
 batchLabel, keepprob:1.})
            gtimg = yMaxOut[0, :, :].reshape(height, width)
            errimg = gtimg - predMaxOut[0, :, :].reshape(height,
 width);
            # Plot
            xs = np.linspace(0, 140, 128); ys = np.linspace(140,
0, 128)
            plt.figure(figsize=(10, 10))
            plt.subplot(2, 2, 1); plt.imshow(refimg); plt.title(
'Input')
            plt.subplot(2, 2, 2); plt.pcolor(xs, ys, gtimg, vmin=
0, vmax=nrclass); plt.title('Ground truth')
            plt.subplot(2, 2, 3); plt.pcolor(xs, ys, predMaxOut[0
, :, :].reshape(height, width), vmin=0, vmax=nrclass); plt.title(
'[Training] Prediction')
            plt.subplot(2, 2, 4); plt.imshow(np.abs(errimg) > 0.5
); plt.title('Error')
            plt.show()
            # Test data
            index = np.random.randint(testData.shape[0])
            batchData = testData[index:index+1]
            batchLabel = testLabelOneHot[index:index+1]
            predMaxOut = sess.run(predmax, feed_dict={x: batchDa
```

```
ta, y: batchLabel, keepprob:1.})
            yMaxOut = sess.run(ymax, feed_dict={x: batchData, y:
 batchLabel, keepprob:1.})
            refimg = testData[index, :, :, :].reshape(height, wi
dth, 3)
            gtimg = yMaxOut[0, :, :].reshape(height, width)
            errimg = gtimg - predMaxOut[0, :, :].reshape(height,
 width)
            # Plot
            plt.figure(figsize=(10, 10))
            plt.subplot(2, 2, 1); plt.imshow(refimg); plt.title(
'Input')
            plt.subplot(2, 2, 2); plt.pcolor(xs, ys, gtimg, vmin=
0, vmax=nrclass);  plt.title('Ground truth')
            plt.subplot(2, 2, 3); plt.pcolor(xs, ys, predMaxOut[0
, :, :].reshape(height, width), vmin=0, vmax=nrclass); plt.title(
'[Validation] Prediction')
            plt.subplot(2, 2, 4); plt.imshow(np.abs(errimg) > 0.5
); plt.title('Error')
            plt.show()

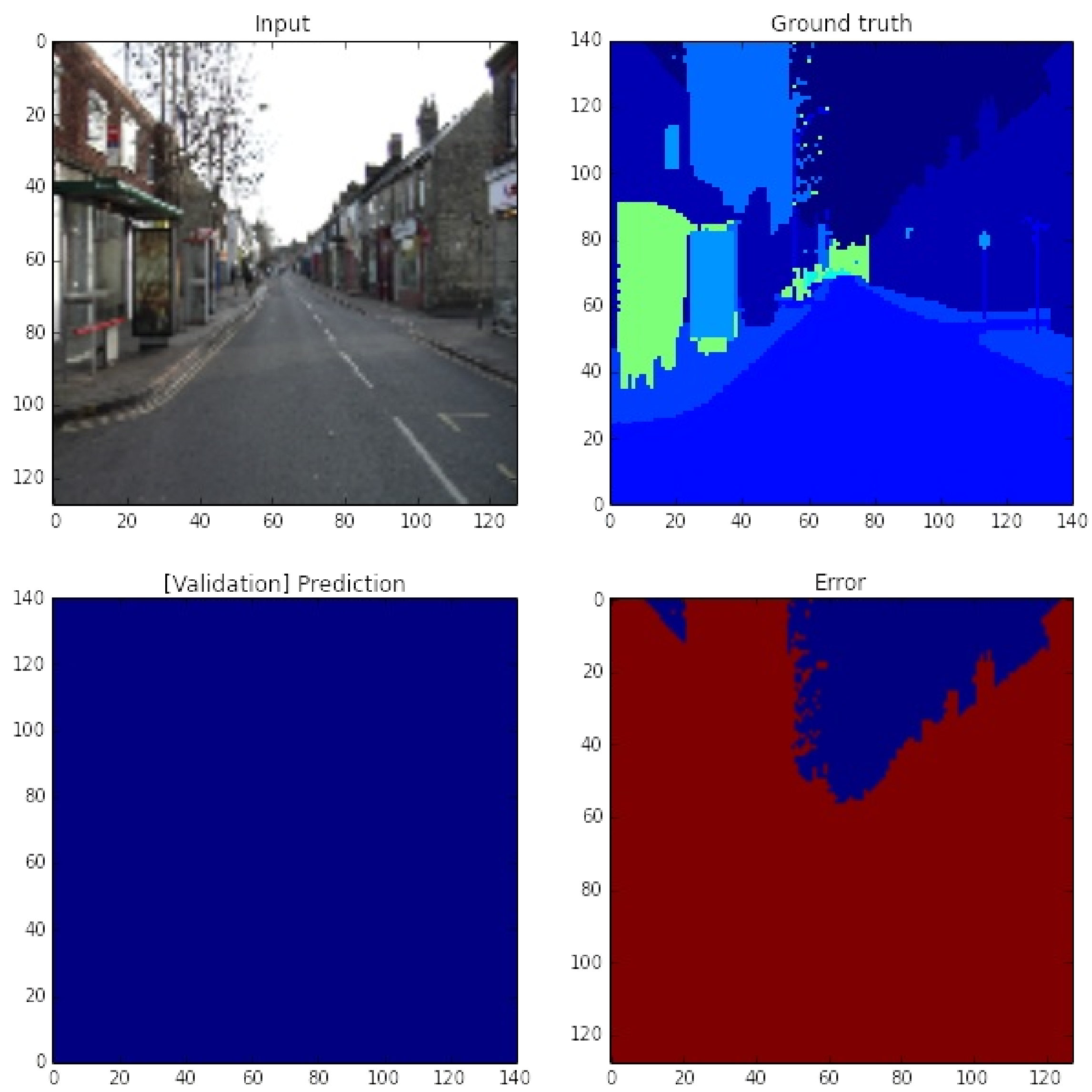
print ("Done")
```

```
checkpoint: None
Couldn't find checkpoint to restore from. Starting over.
[00/1000] trainLoss: nan trainAcc: 0.18 valLoss: nan valAcc: 0.1
7


/usr/lib/pymodules/python2.7/matplotlib/collections.py:548: Futu
reWarning: elementwise comparison failed; returning scalar inste
ad, but in the future will perform elementwise comparison
  if self._edgecolors == 'face':
```



Input
Ground truth
[Training] Prediction
Error



Input
Ground truth
[Validation] Prediction
Error

```
import matplotlib.pyplot as plt
import numpy as np
import math
from PIL import Image
import cPickle as pkl
import time
import tensorflow as tf
import tensorflow.examples.tutorials.mnist.input_data as input_d
ata
import glob
%matplotlib inline
print ("Packages loaded")
```

```
Packages loaded
```

Load dataset

```
dirpath = "./data/iccv09Data/images/"
height = 240
width  = 320
resize_ratio = 3
nr_img = 0
fileList = glob.glob(dirpath + '*.jpg')
for i, file in enumerate(fileList):
    img = Image.open(file)
    array = np.array(img)
    if array.shape[0] == height and array.shape[1] == width:
        nr_img = nr_img + 1
        rgb = array.reshape(1, height, width, 3)
        imglow = img.resize((int(width/resize_ratio)
                ,int(height/resize_ratio)), Image.BICUBIC)
        imglow = imglow.resize((width, height), Image.BICUBIC)
        rgblow = np.array(np.float32(imglow)/255.)
        rgblow = rgblow.reshape(1, height, width, 3)
        rgb = np.reshape(rgb, [1, -1])
        rgblow = np.reshape(rgblow, [1, -1])
        if nr_img == 1:
            data = rgb
            datalow = rgblow
        else:
            data = np.concatenate((data, rgb), axis=0)
            datalow = np.concatenate((datalow, rgblow), axis=0)

print ("nr_img is %d" % (nr_img))
print ("Shape of 'data' is %s" % (data.shape,))
print ("Shape of 'datalow' is %s" % (datalow.shape,))
```

```
nr_img is 531
Shape of 'data' is (531, 230400)
Shape of 'datalow' is (531, 230400)
```

Divide into two sets

(xtrain, ytrain) and (xtest, ytest)

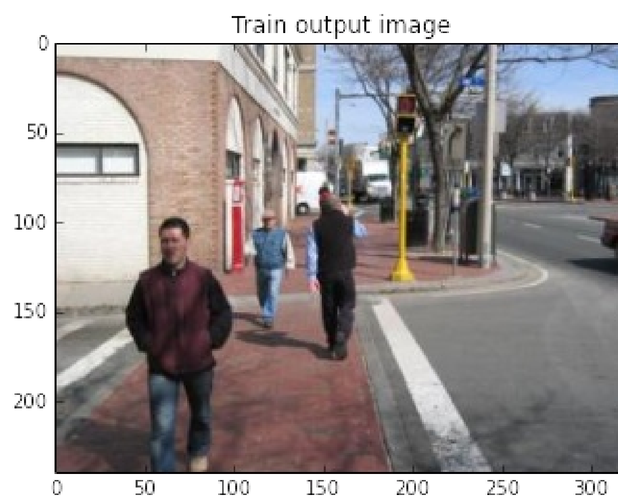
```
randidx = np.random.permutation(nr_img)
nrtrain = int(nr_img*0.7)
nrtest  = nr_img - nrtrain
xtrain  = datalow[randidx[0:nrtrain], :]
ytrain  = data[randidx[0:nrtrain], :]
xtest   = datalow[randidx[nrtrain:nr_img], :]
ytest   = data[randidx[nrtrain:nr_img], :]
print ("Shape of 'xtrain' is %s" % (xtrain.shape,))
print ("Shape of 'ytrain' is %s" % (ytrain.shape,))
print ("Shape of 'xtest' is %s" % (xtest.shape,))
print ("Shape of 'ytest' is %s" % (ytest.shape,))
```

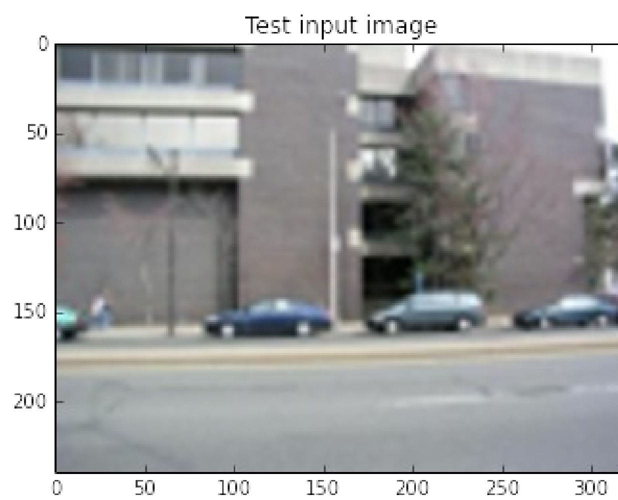
```
Shape of 'xtrain' is (371, 230400)
Shape of 'ytrain' is (371, 230400)
Shape of 'xtest' is (160, 230400)
Shape of 'ytest' is (160, 230400)
```

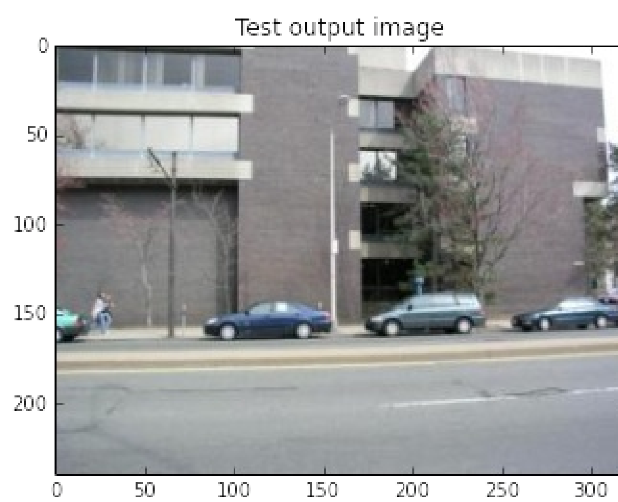
Plot some images

```
# Train
randidx = np.random.randint(nrtrain)
currx = xtrain[randidx, :]
currx = np.reshape(currx, [height, width, 3])
plt.imshow(currx)
plt.title("Train input image")
plt.show()
curry = ytrain[randidx, :]
curry = np.reshape(curry, [height, width, 3])
plt.imshow(curry)
plt.title("Train output image")
plt.show()
# Test
randidx = np.random.randint(nrtest)
currx = xtest[randidx, :]
currx = np.reshape(currx, [height, width, 3])
plt.imshow(currx)
plt.title("Test input image")
plt.show()
curry = ytest[randidx, :]
curry = np.reshape(curry, [height, width, 3])
plt.imshow(curry)
plt.title("Test output image")
plt.show()
```









Define network

```
n1 = 32
n2 = 64
n3 = 64
n4 = 64
```

```
n5 = 64
n6 = 3
ksize = 3
weights = {
    'ce1': tf.Variable(tf.random_normal([ksize, ksize, 3, n1],
stddev=0.01)),
    'ce2': tf.Variable(tf.random_normal([ksize, ksize, n1, n2],
stddev=0.01)),
    'ce3': tf.Variable(tf.random_normal([ksize, ksize, n2, n3],
stddev=0.01)),
    'ce4': tf.Variable(tf.random_normal([ksize, ksize, n3, n4],
stddev=0.01)),
    'ce5': tf.Variable(tf.random_normal([ksize, ksize, n4, n5],
stddev=0.01)),
    'ce6': tf.Variable(tf.random_normal([ksize, ksize, n5, n6],
 stddev=0.01))
}
biases = {
    'be1': tf.Variable(tf.random_normal([n1], stddev=0.01)),
    'be2': tf.Variable(tf.random_normal([n2], stddev=0.01)),
    'be3': tf.Variable(tf.random_normal([n3], stddev=0.01)),
    'be4': tf.Variable(tf.random_normal([n4], stddev=0.01)),
    'be5': tf.Variable(tf.random_normal([n5], stddev=0.01)),
    'be6': tf.Variable(tf.random_normal([n6], stddev=0.01))
}
def srn(_X, _W, _b, _keepprob):
    _input_r = tf.reshape(_X, shape=[-1, height, width, 3])
    # Encoder
    _ce1 = tf.nn.relu(tf.add(tf.nn.conv2d(_input_r, _W['ce1']
        , strides=[1, 1, 1, 1], padding='SAME'), _b['be1']))
    _ce1 = tf.nn.dropout(_ce1, _keepprob)
    _ce2 = tf.nn.relu(tf.add(tf.nn.conv2d(_ce1, _W['ce2']
        , strides=[1, 1, 1, 1], padding='SAME'), _b['be2']))
    _ce2 = tf.nn.dropout(_ce2, _keepprob)
    _ce3 = tf.nn.relu(tf.add(tf.nn.conv2d(_ce2, _W['ce3']
        , strides=[1, 1, 1, 1], padding='SAME'), _b['be3']))
    _ce3 = tf.nn.dropout(_ce3, _keepprob)

    _ce4 = tf.nn.relu(tf.add(tf.nn.conv2d(_ce3, _W['ce4']
        , strides=[1, 1, 1, 1], padding='SAME'), _b['be4']))
    _ce4 = tf.nn.dropout(_ce4, _keepprob)
    _ce5 = tf.nn.relu(tf.add(tf.nn.conv2d(_ce4, _W['ce5']
        , strides=[1, 1, 1, 1], padding='SAME'), _b['be5']))
    _ce5 = tf.nn.dropout(_ce5, _keepprob)
    _ce6 = tf.nn.relu(tf.add(tf.nn.conv2d(_ce5, _W['ce6']
        , strides=[1, 1, 1, 1], padding='SAME'), _b['be6']))
    _out = _ce6 + _input_r
    return {'input_r': _input_r, 'ce1': _ce1, 'ce2': _ce2, 'ce3'
: _ce3
        , 'ce4': _ce4, 'ce5': _ce5, 'ce6': _ce6
        , 'layers': (_input_r, _ce1, _ce2, _ce3, _ce4, _ce5, _ce
6)
        , 'out': _out}
```

```
print ("Network ready")
```

```
Network ready
```

Define functions

```
dim = height*width*3
x = tf.placeholder(tf.float32, [None, dim])
y = tf.placeholder(tf.float32, [None, dim])
keepprob = tf.placeholder(tf.float32)
pred = srn(x, weights, biases, keepprob)['out']
cost = tf.reduce_mean(tf.square(srn(x, weights, biases, keepprob
)['out']
            - tf.reshape(y, shape=[-1, height, width, 3])))
learning_rate = 0.001
optm = tf.train.AdamOptimizer(learning_rate, 0.9).minimize(cost)
init = tf.initialize_all_variables()
print ("Functions ready")
```

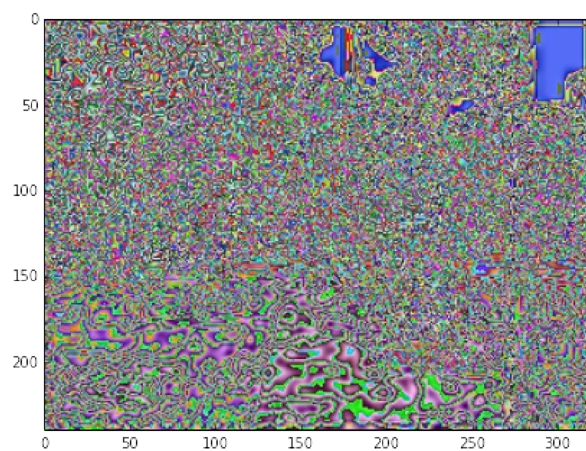
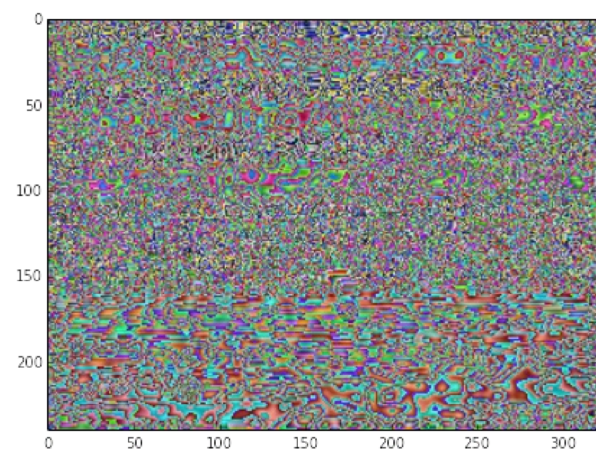
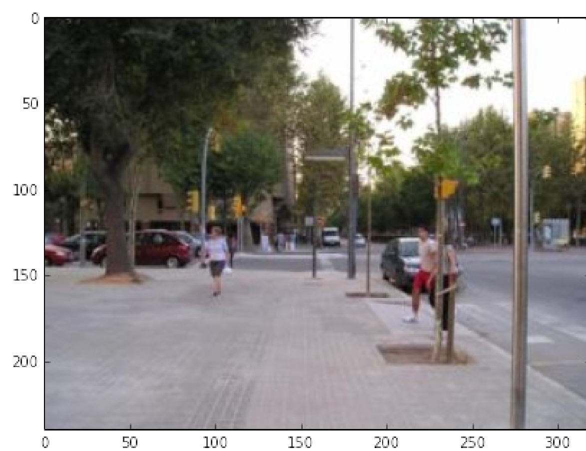
```
Functions ready
```

Run

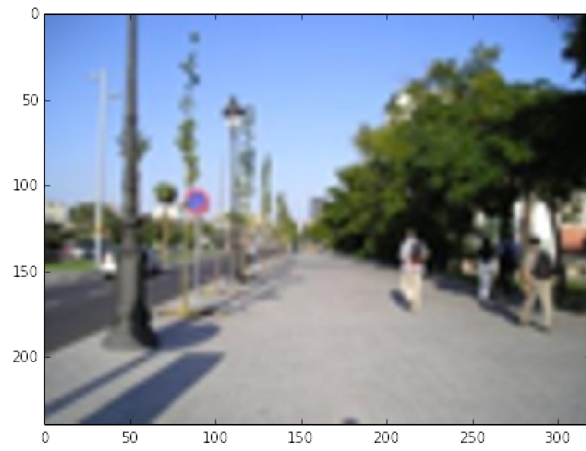
```
sess = tf.Session()
sess.run(init)
# Fit all training data
batch_size = 16
n_epochs   = 100000
print("Strart training..")
for epoch_i in range(n_epochs):
    for batch_i in range(nrtrain // batch_size):
        randidx = np.random.randint(nrtrain, size=batch_size)
        batch_xs = xtrain[randidx, :]
        batch_ys = ytrain[randidx, :]
        sess.run(optm, feed_dict={x: batch_xs
            , y: batch_ys, keepprob: 0.7})
    if (epoch_i % 10) == 0:
        print ("[%02d/%02d] cost: %.4f" % (epoch_i, n_epochs
            , sess.run(cost, feed_dict={x: batch_xs
            , y: batch_ys, keepprob: 1.})))
    if (epoch_i % 100) == 0:
        n_examples = 2
        print ("Training dataset")
        randidx = np.random.randint(nrtrain, size=n_examples)
        train_xs = xtrain[randidx, :]
        train_ys = ytrain[randidx, :]
        recon = sess.run(pred, feed_dict={x: train_xs, keepprob:
1.})
        fig, axs = plt.subplots(3, n_examples, figsize=(15, 20))
        for example_i in range(n_examples):
            axs[0][example_i].imshow(np.reshape(
                train_xs[example_i, :], (height, width, 3)))
            axs[1][example_i].imshow(np.reshape(
                recon[example_i, :], (height, width, 3)))
            axs[2][example_i].imshow(np.reshape(
                train_ys[example_i, :], (height, width, 3)))
        plt.show()
        print ("Test dataset")
        randidx = np.random.randint(nrtest, size=n_examples)
        test_xs = xtest[randidx, :]
        test_ys = ytest[randidx, :]
        recon = sess.run(pred, feed_dict={x: test_xs, keepprob:
1.})
        fig, axs = plt.subplots(3, n_examples, figsize=(15, 20))
        for example_i in range(n_examples):
            axs[0][example_i].imshow(np.reshape(
                test_xs[example_i, :], (height, width, 3)))
            axs[1][example_i].imshow(np.reshape(
                recon[example_i, :], (height, width, 3)))
            axs[2][example_i].imshow(np.reshape(
                test_ys[example_i, :], (height, width, 3)))
        plt.show()
print("Training done. ")
```

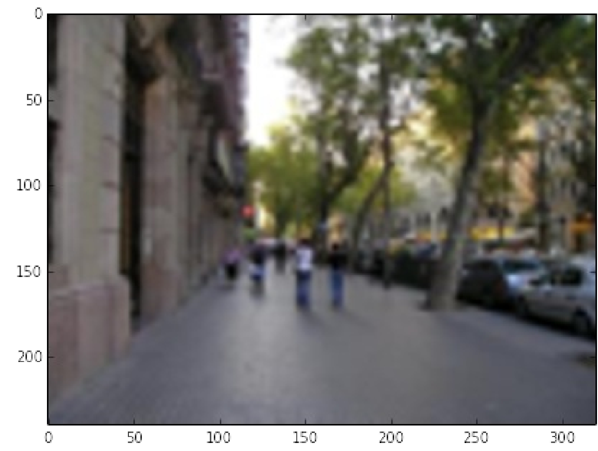
```
Strart training..
[00/100000] cost: 3772.8538
Training dataset
```

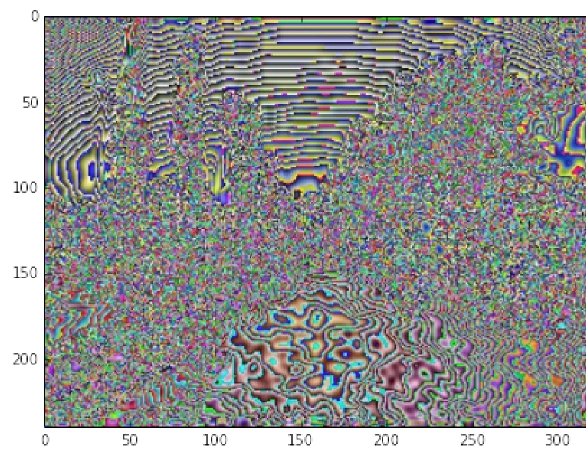


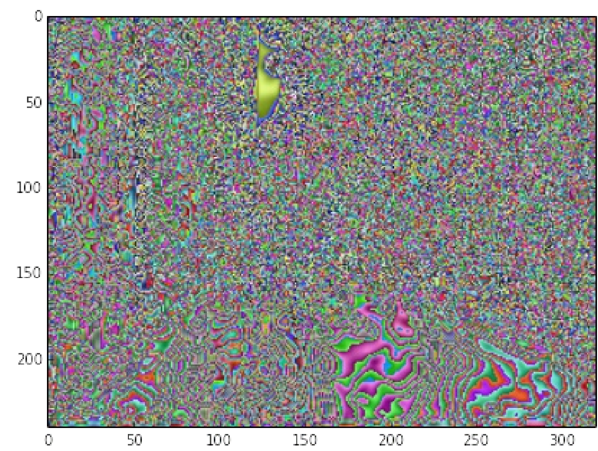





```
Test dataset
```

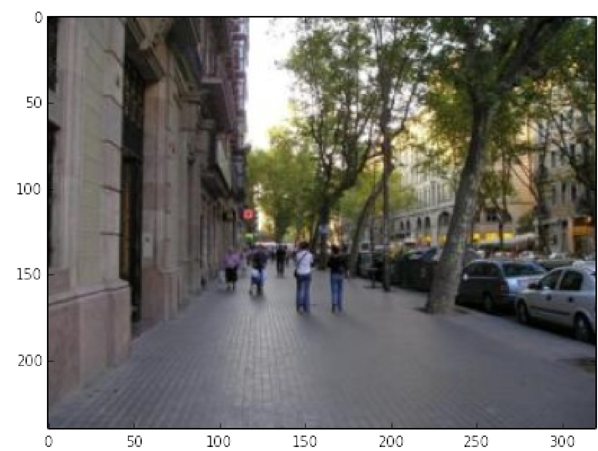




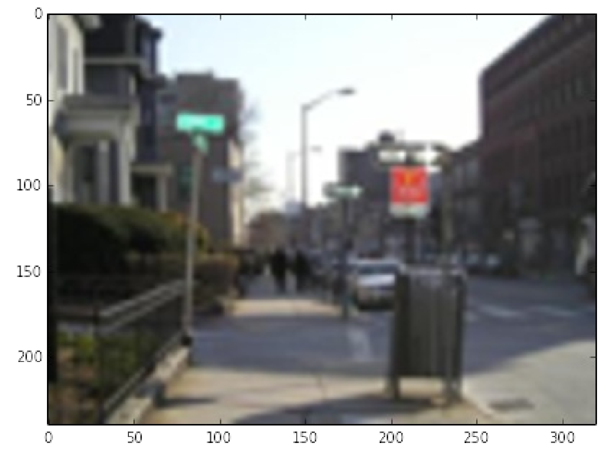
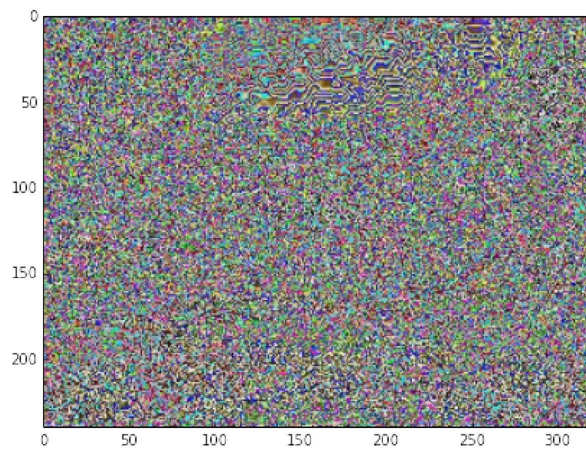
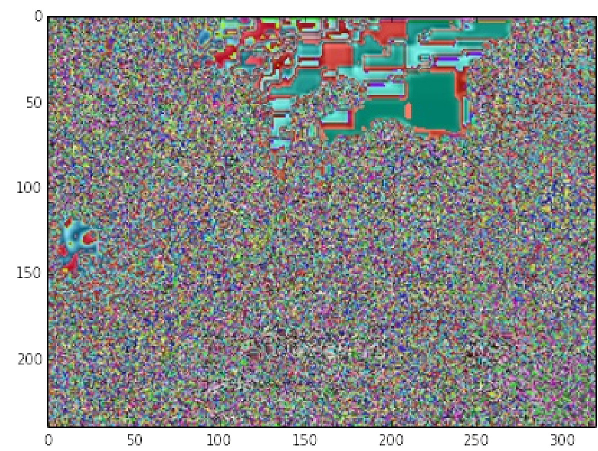








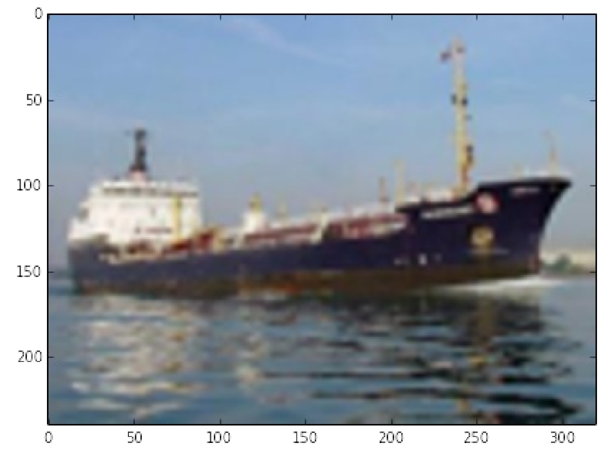
```
[10/100000] cost: 702.8953
[20/100000] cost: 506.8691
[30/100000] cost: 532.7783
[40/100000] cost: 459.9507
[50/100000] cost: 392.8480
[60/100000] cost: 415.6570
[70/100000] cost: 354.2756
[80/100000] cost: 357.1342
[90/100000] cost: 320.3797
[100/100000] cost: 440.0826
Training dataset
```

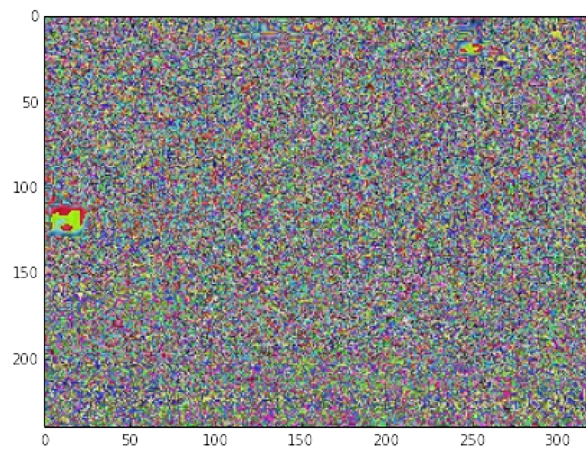





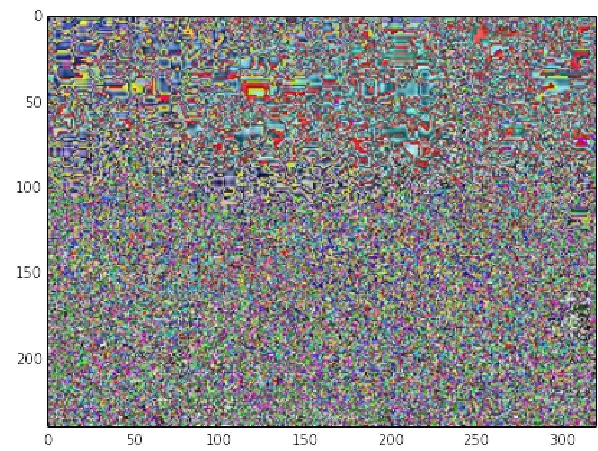


```
Test dataset
```


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300

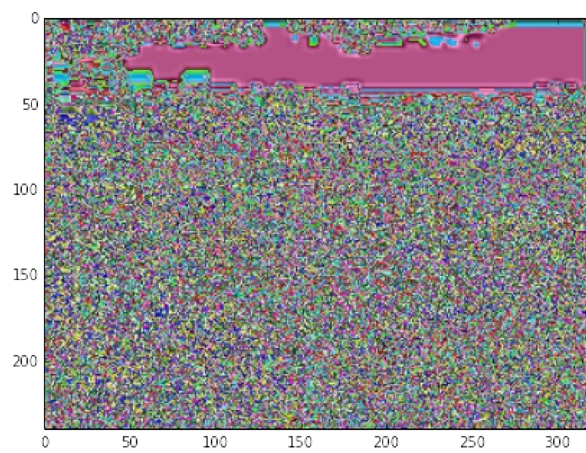
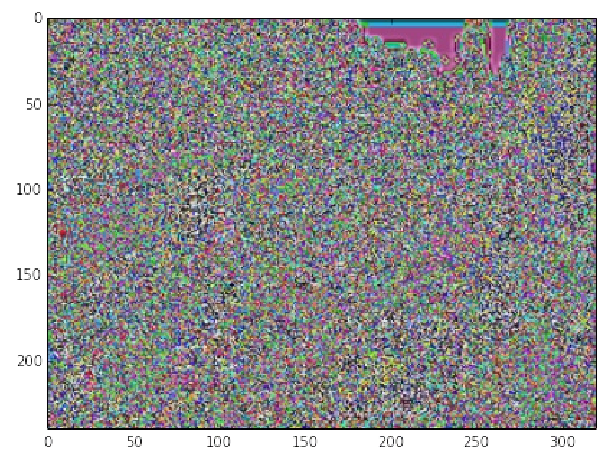

0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300

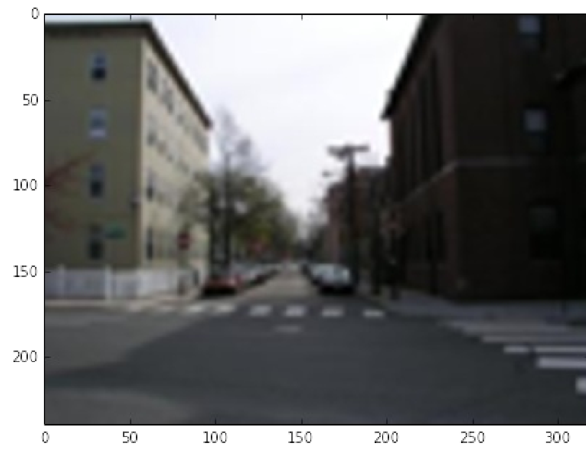
```
[110/100000] cost: 324.7620
[120/100000] cost: 326.6705
[130/100000] cost: 434.0658
[140/100000] cost: 446.6097
[150/100000] cost: 411.9198
[160/100000] cost: 501.3135
[170/100000] cost: 433.7086
[180/100000] cost: 400.1469
[190/100000] cost: 330.6881
[200/100000] cost: 522.5967
Training dataset
```

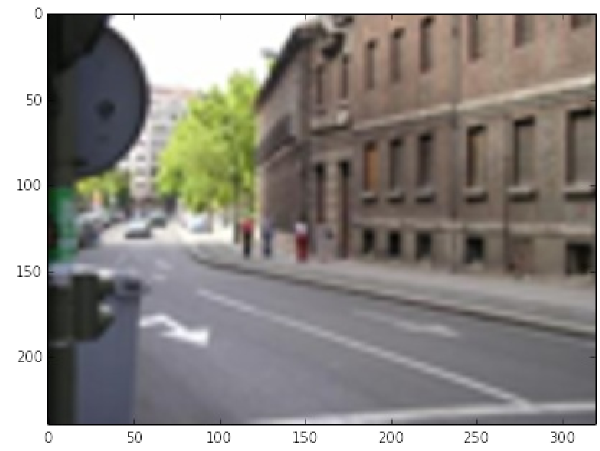


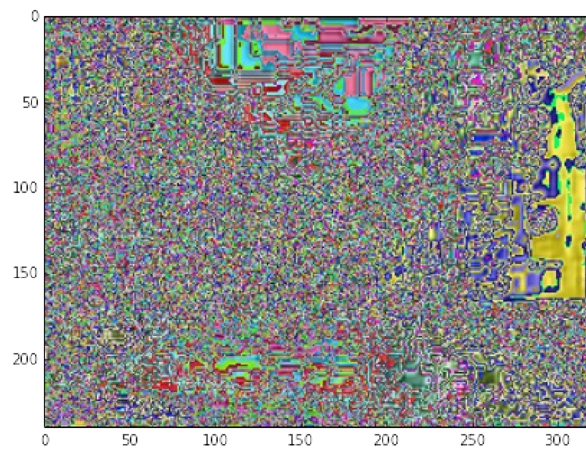



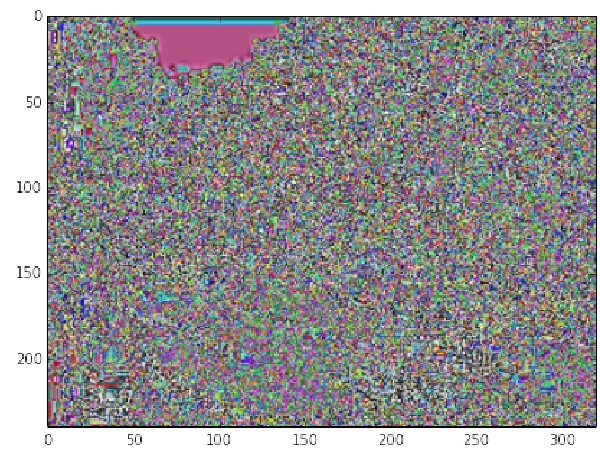


```
Test dataset
```


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300

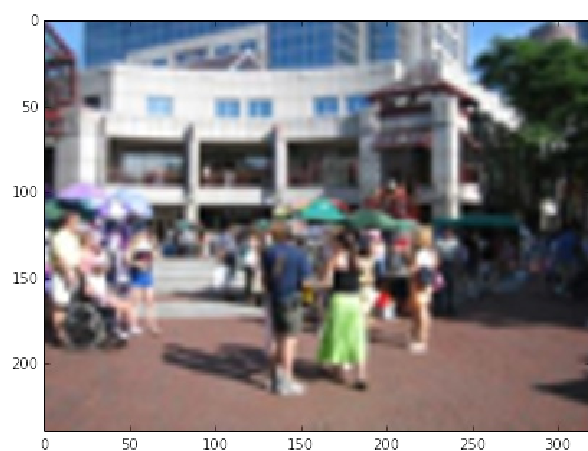

0
50
100
150
200
0 50 100 150 200 250 300

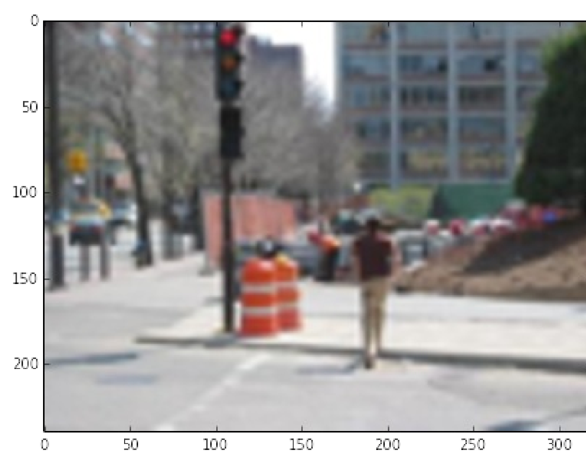

0
50
100
150
200
0 50 100 150 200 250 300

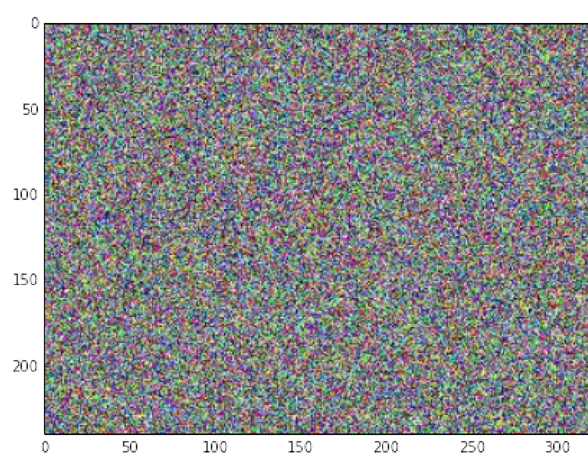

0
50
100
150
200
0 50 100 150 200 250 300

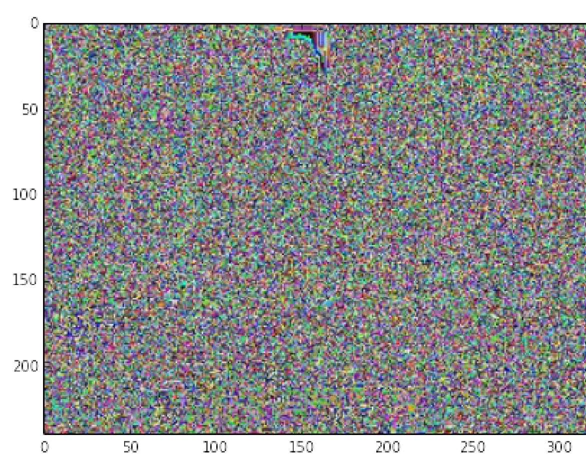

0
50
100
150
200
0 50 100 150 200 250 300

```
[210/100000] cost: 486.0645
[220/100000] cost: 410.2568
[230/100000] cost: 428.8757
[240/100000] cost: 552.0220
[250/100000] cost: 422.1021
[260/100000] cost: 377.5060
[270/100000] cost: 269.3975
[280/100000] cost: 219.0566
[290/100000] cost: 216.6913
[300/100000] cost: 248.9276
Training dataset
```









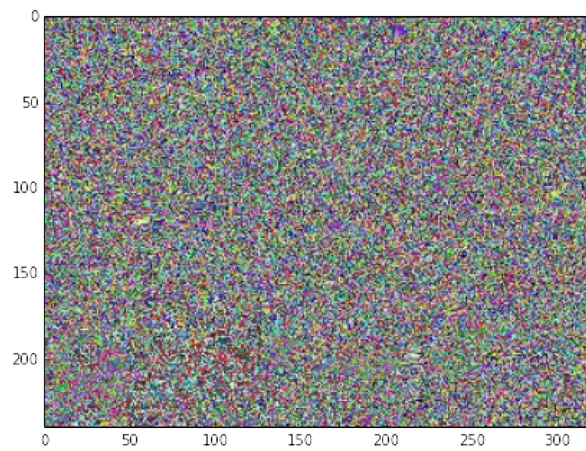


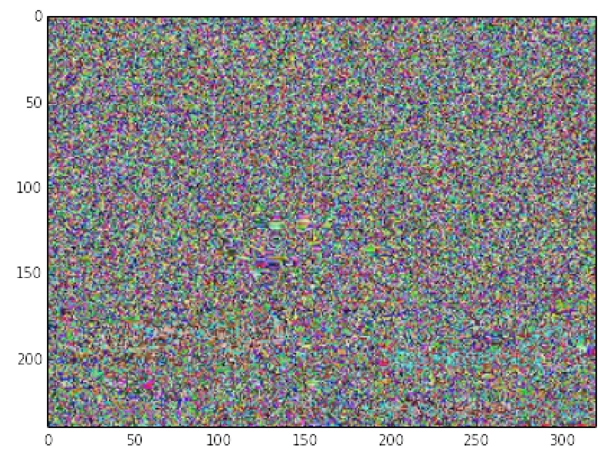


```
Test dataset
```


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300

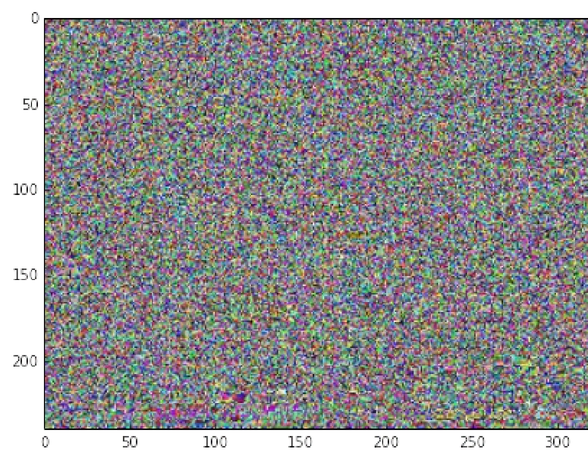

0
50
100
150
200
0 50 100 150 200 250 300

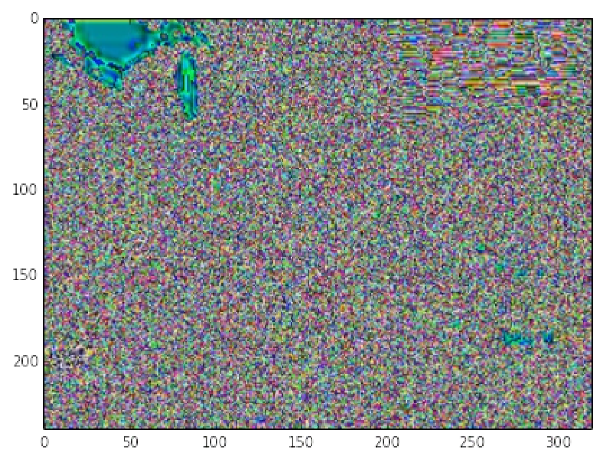

0
50
100
150
200
0 50 100 150 200 250 300

```
[310/100000] cost: 230.1701
[320/100000] cost: 254.1594
[330/100000] cost: 233.3717
[340/100000] cost: 177.7164
[350/100000] cost: 203.6235
[360/100000] cost: 256.6276
[370/100000] cost: 230.0090
[380/100000] cost: 219.8041
[390/100000] cost: 215.0632
[400/100000] cost: 212.2820
Training dataset
```









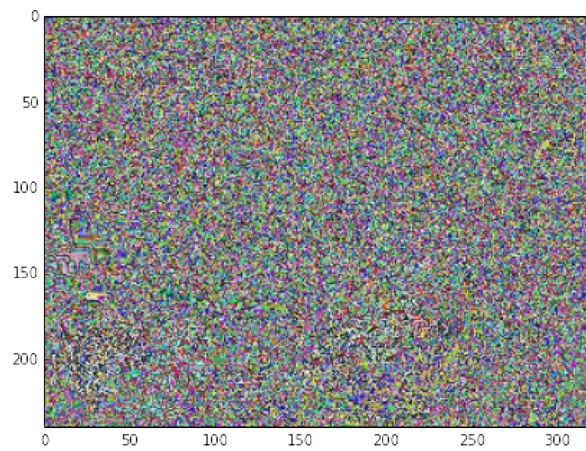


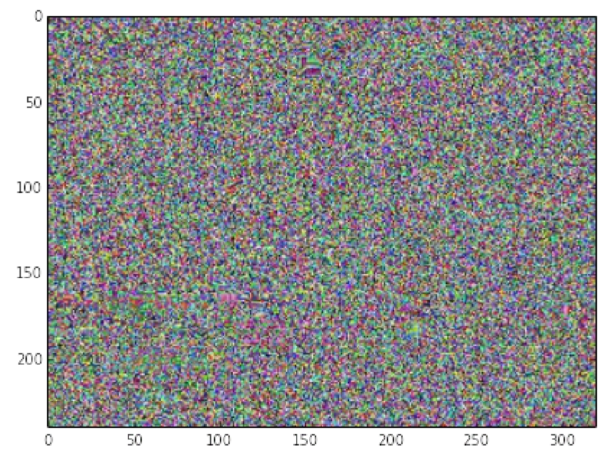



```
Test dataset
```


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300
STOP


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300

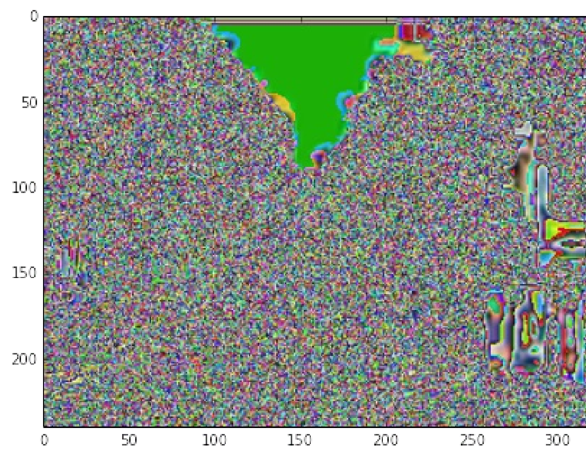

0
50
100
150
200
0 50 100 150 200 250 300

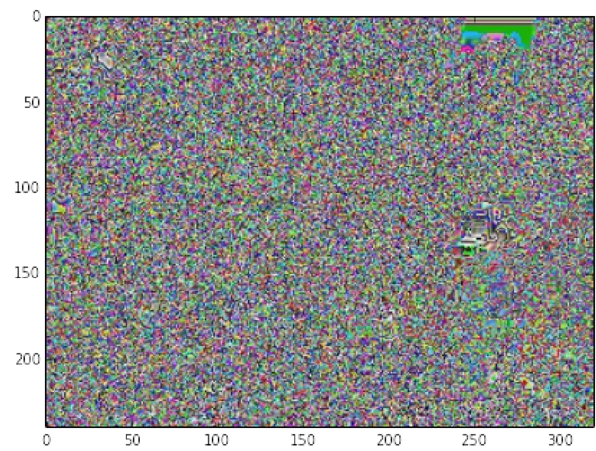

0
50
100
150
200
0 50 100 150 200 250 300
ONE WAY
STOP

```
[410/100000] cost: 198.7617
[420/100000] cost: 196.8098
[430/100000] cost: 219.6303
[440/100000] cost: 171.0177
[450/100000] cost: 215.5296
[460/100000] cost: 177.6324
[470/100000] cost: 190.9361
[480/100000] cost: 201.9544
[490/100000] cost: 191.6286
[500/100000] cost: 200.1959
Training dataset
```





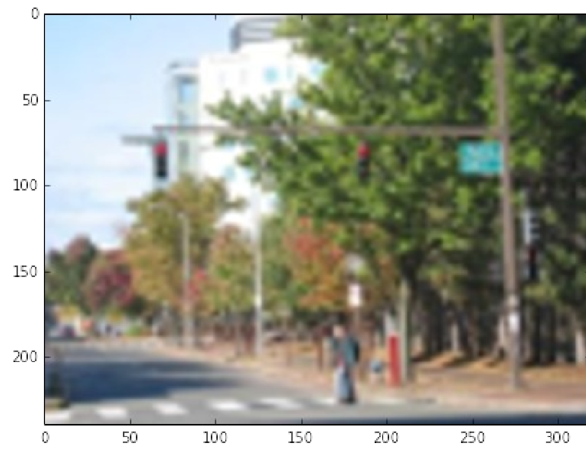


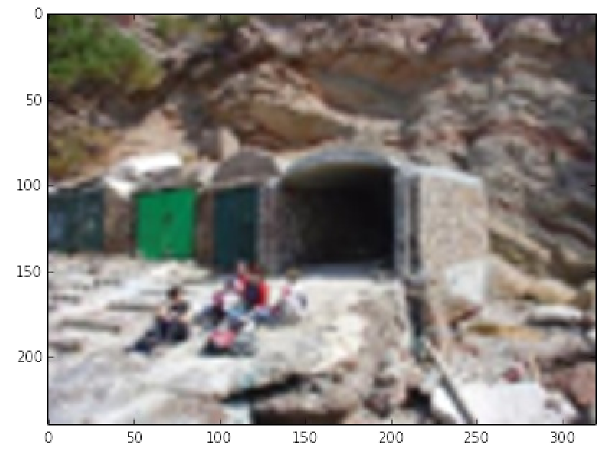


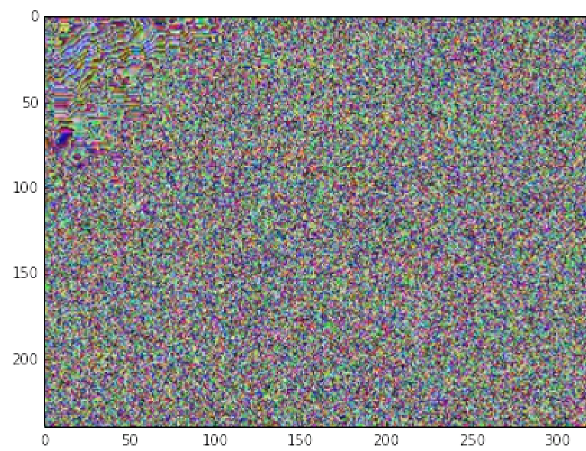


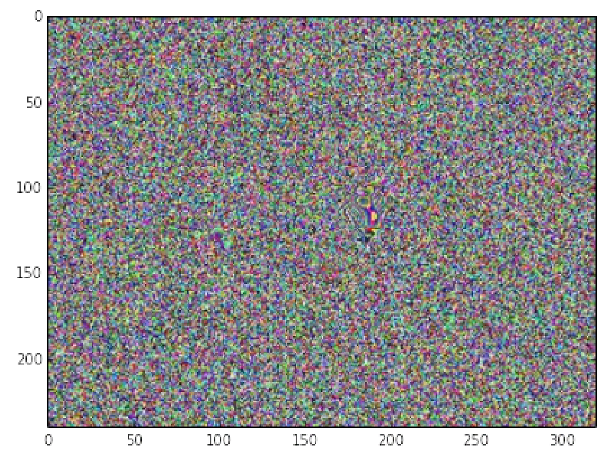


```
Test dataset
```


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300

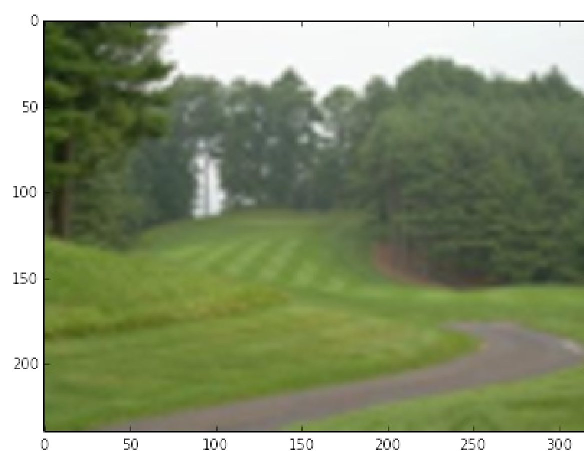
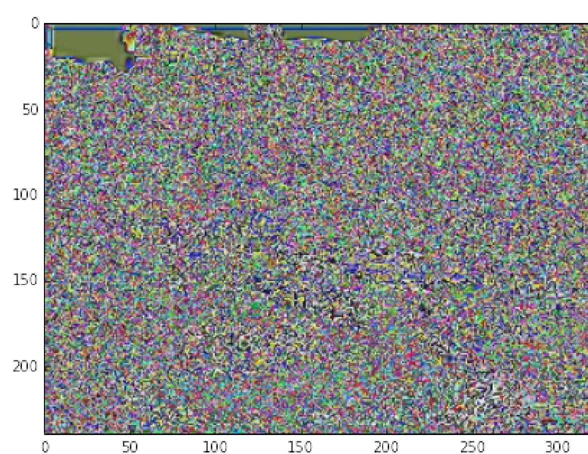
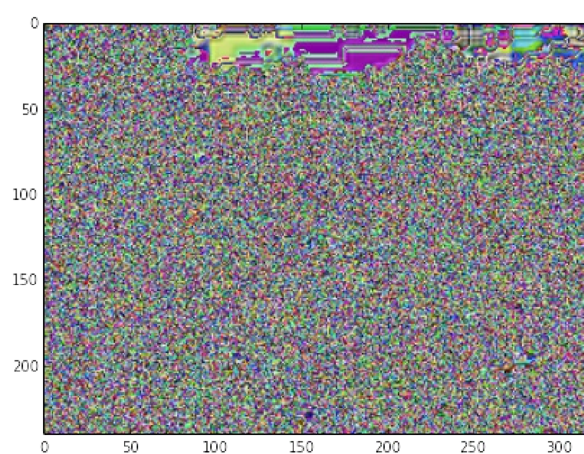

0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300

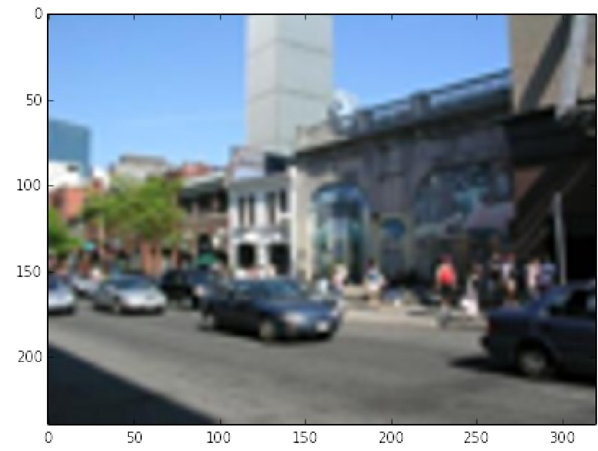
```
[510/100000] cost: 199.5182
[520/100000] cost: 170.4744
[530/100000] cost: 197.0104
[540/100000] cost: 202.3129
[550/100000] cost: 225.0957
[560/100000] cost: 179.6629
[570/100000] cost: 190.9138
[580/100000] cost: 177.1373
[590/100000] cost: 230.7214
[600/100000] cost: 199.4890
Training dataset
```

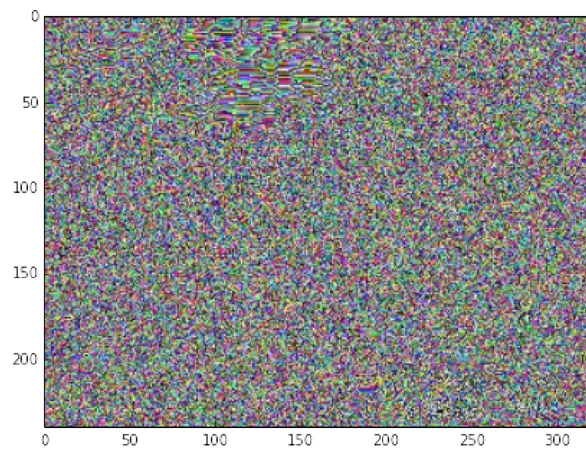





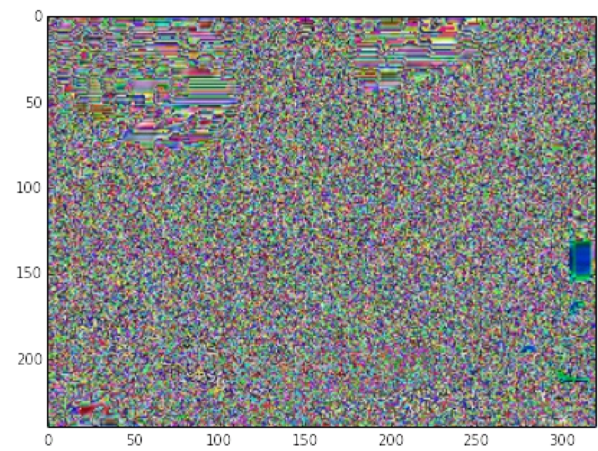


```
Test dataset
```


0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

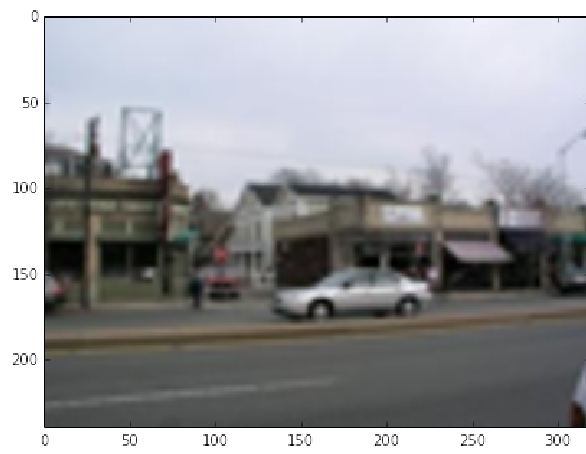
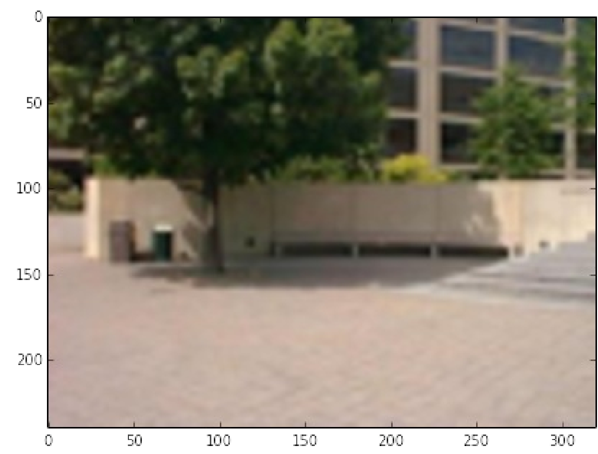
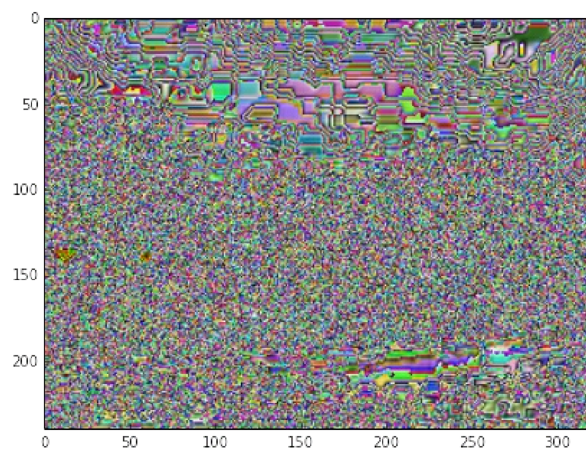
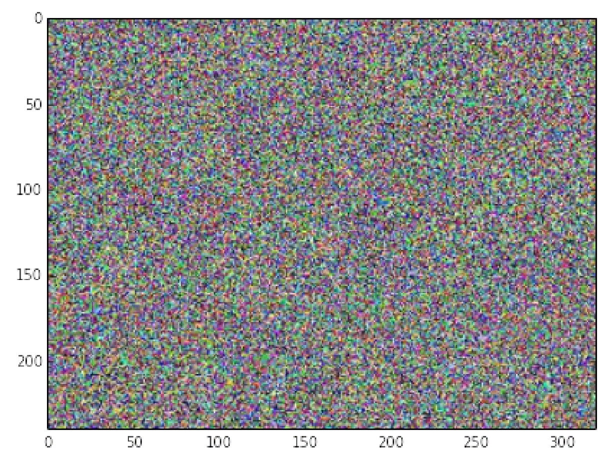

0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

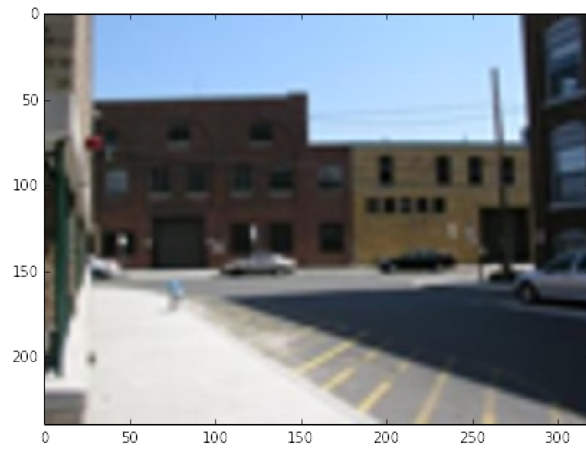

0
50
100
150
200
0
50
100
150
200
250
300

```
[610/100000] cost: 150.3460
[620/100000] cost: 142.2733
[630/100000] cost: 184.0573
[640/100000] cost: 220.4450
[650/100000] cost: 165.4367
[660/100000] cost: 253.1218
[670/100000] cost: 158.5751
[680/100000] cost: 227.8497
[690/100000] cost: 167.6208
[700/100000] cost: 196.1220
Training dataset
```

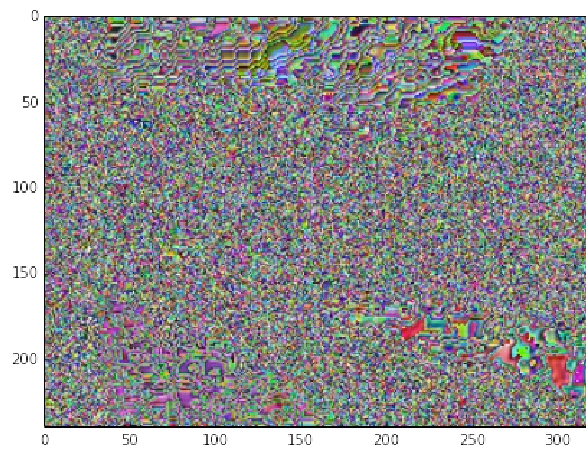





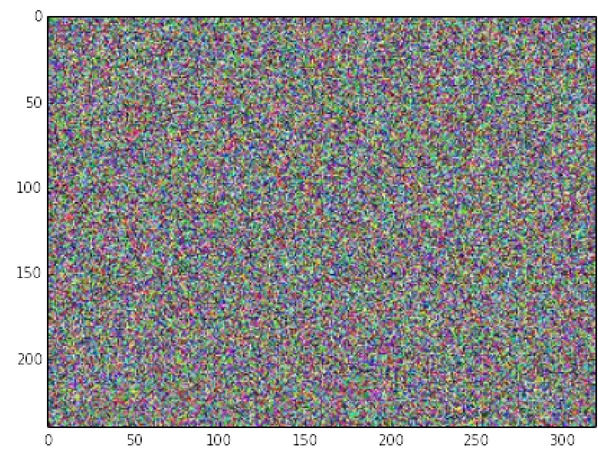


```
Test dataset
```


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300

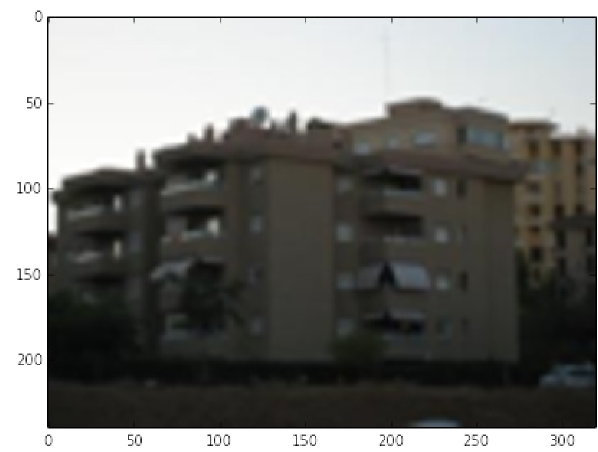
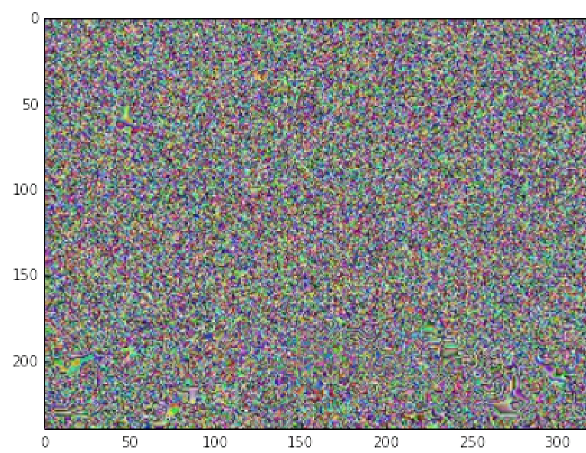
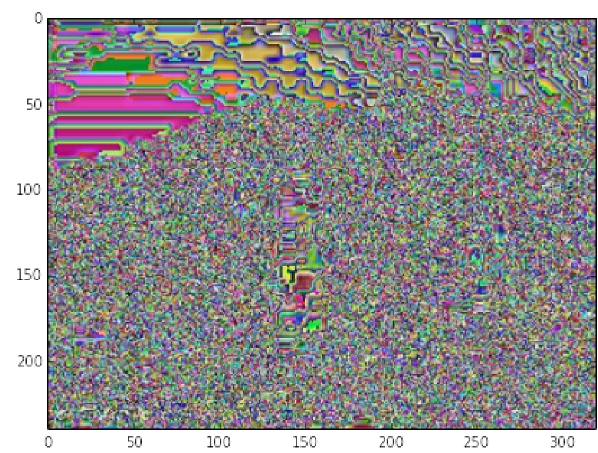

0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300

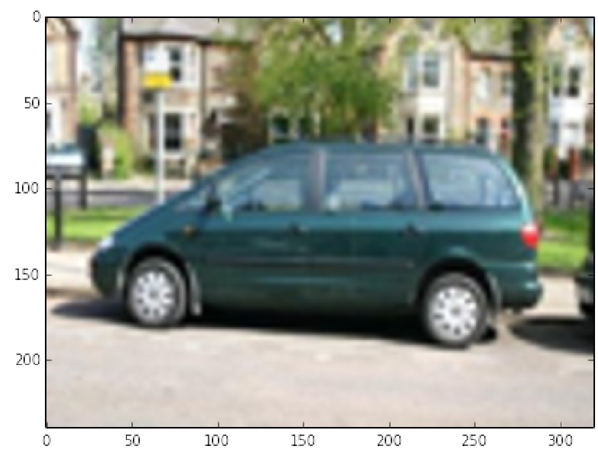
```
[710/100000] cost: 197.5616
[720/100000] cost: 198.3330
[730/100000] cost: 200.2553
[740/100000] cost: 204.8952
[750/100000] cost: 209.1677
[760/100000] cost: 205.5825
[770/100000] cost: 158.0817
[780/100000] cost: 205.6169
[790/100000] cost: 188.0784
[800/100000] cost: 196.0623
Training dataset
```

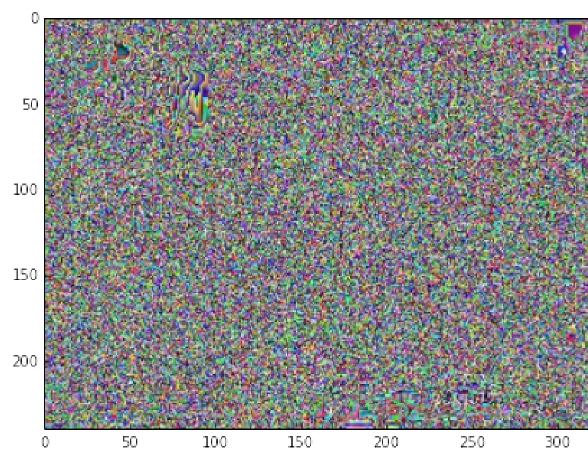





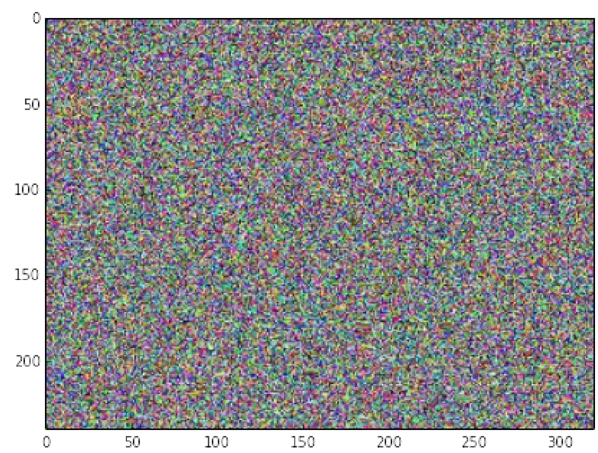


```
Test dataset
```

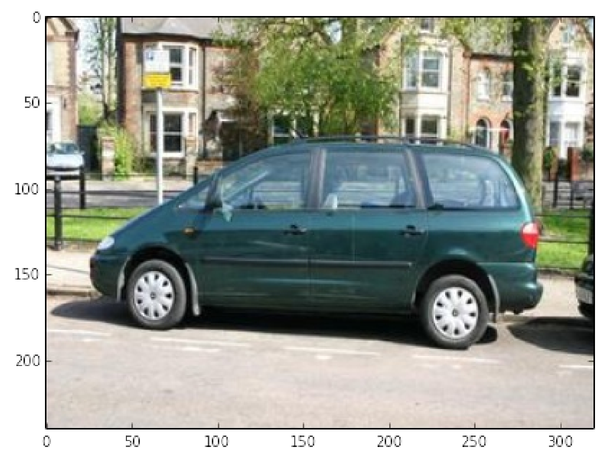

0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300

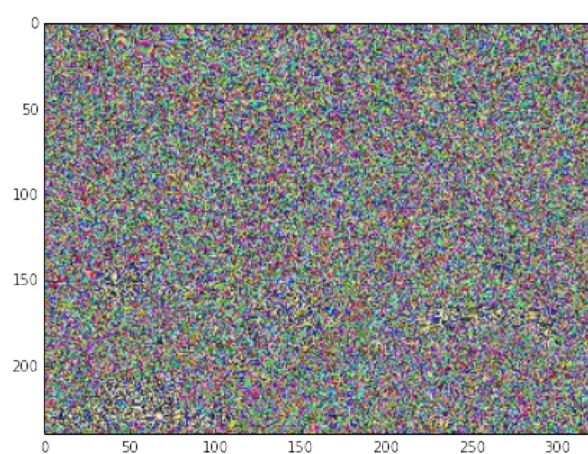

0
50
100
150
200
0 50 100 150 200 250 300

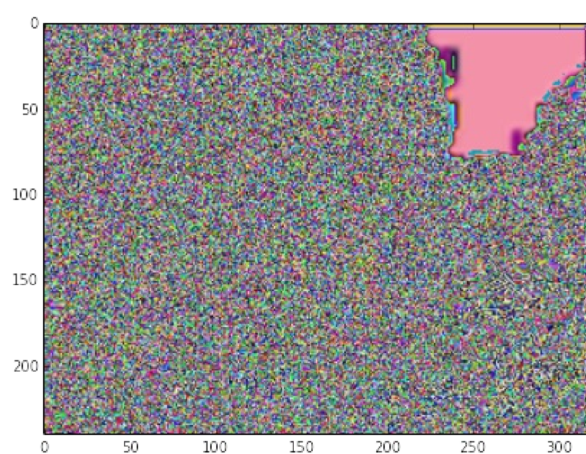

0
50
100
150
200
0 50 100 150 200 250 300

```
[810/100000] cost: 210.1583
[820/100000] cost: 221.4599
[830/100000] cost: 237.4969
[840/100000] cost: 192.0751
[850/100000] cost: 213.0567
[860/100000] cost: 174.9142
[870/100000] cost: 179.2275
[880/100000] cost: 211.8621
[890/100000] cost: 212.8904
[900/100000] cost: 257.3144
Training dataset
```







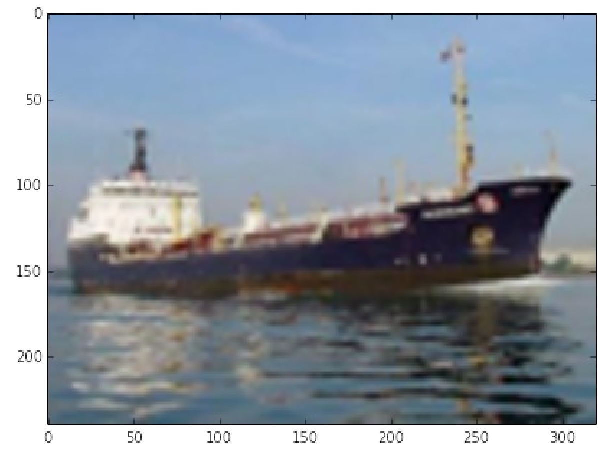


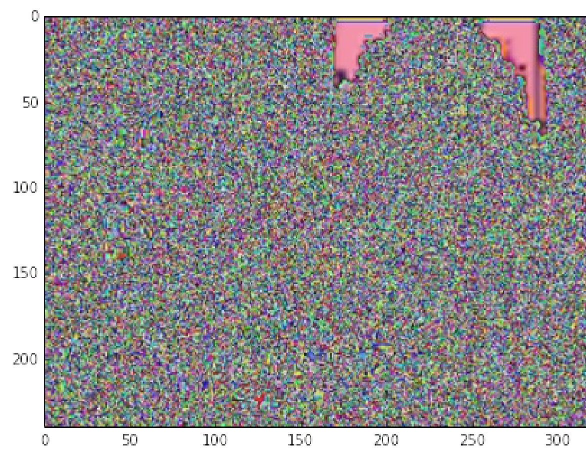


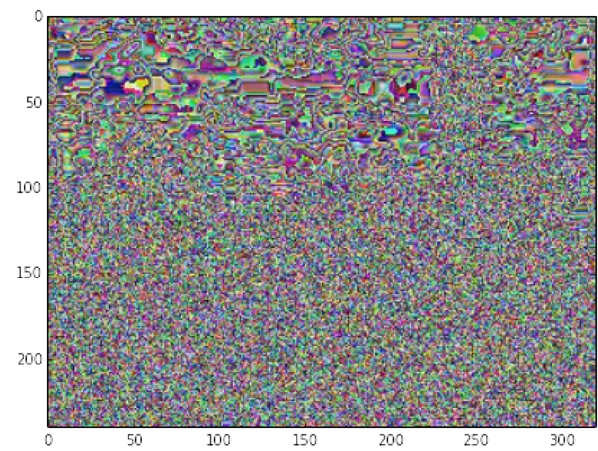



```
Test dataset
```


0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

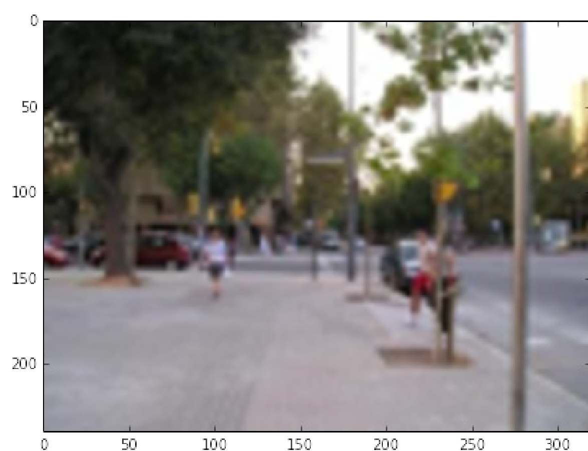

0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

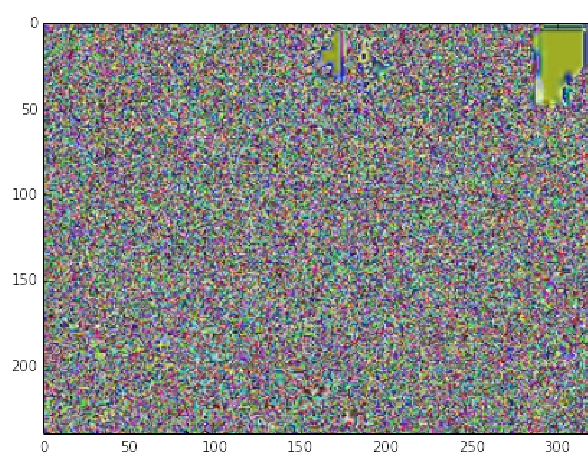

0
50
100
150
200
0
50
100
150
200
250
300

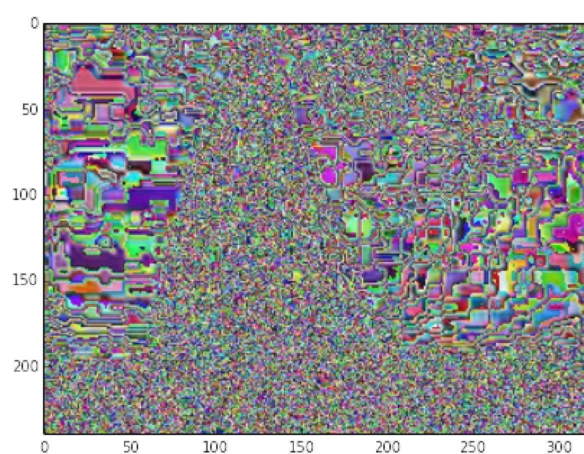

0
50
100
150
200
0
50
100
150
200
250
300

```
[910/100000] cost: 279.0889
[920/100000] cost: 275.9886
[930/100000] cost: 294.9624
[940/100000] cost: 218.6249
[950/100000] cost: 200.8324
[960/100000] cost: 287.3336
[970/100000] cost: 245.9705
[980/100000] cost: 240.9357
[990/100000] cost: 223.2813
[1000/100000] cost: 258.9540
Training dataset
```





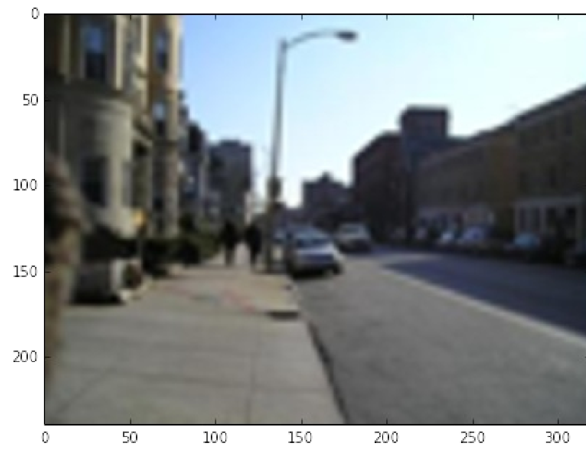




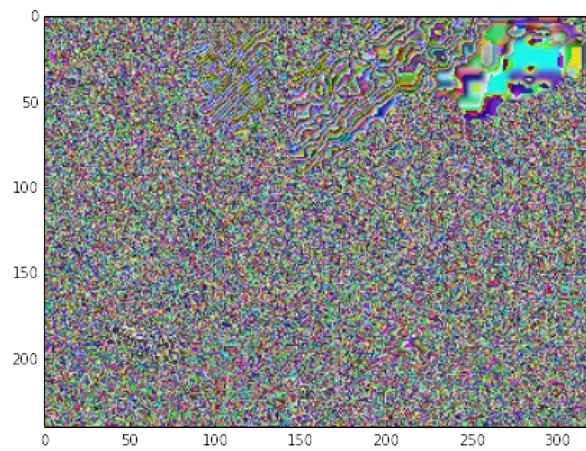


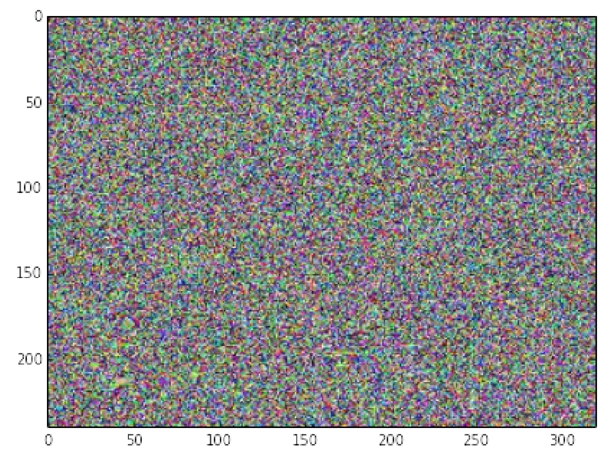



```
Test dataset
```


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300


0
50
100
150
200
0 50 100 150 200 250 300

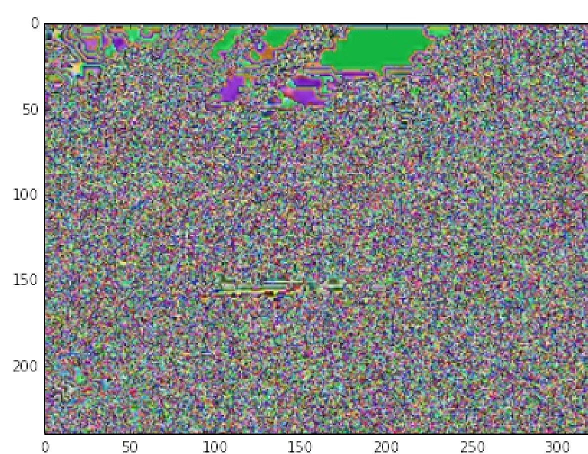

0
50
100
150
200
0 50 100 150 200 250 300

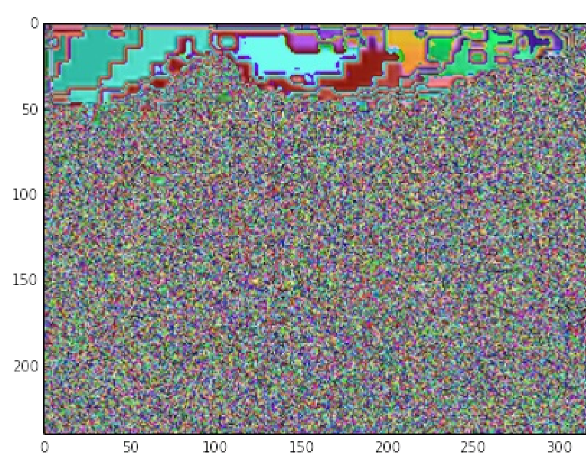

0
50
100
150
200
0 50 100 150 200 250 300

```
[1010/100000] cost: 270.5289
[1020/100000] cost: 271.8137
[1030/100000] cost: 246.7184
[1040/100000] cost: 192.9518
[1050/100000] cost: 222.2798
[1060/100000] cost: 248.5685
[1070/100000] cost: 234.6805
[1080/100000] cost: 203.7659
[1090/100000] cost: 253.0008
[1100/100000] cost: 269.9266
Training dataset
```





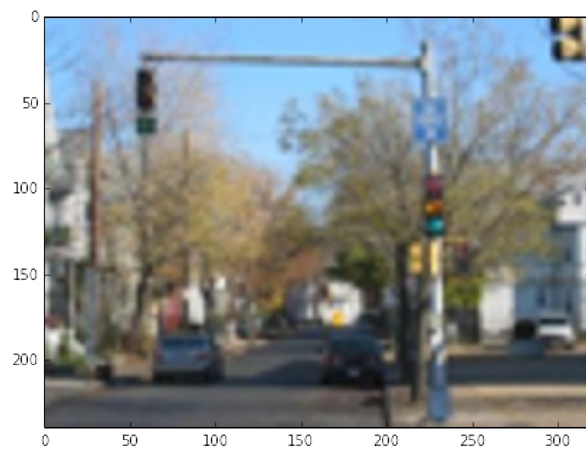




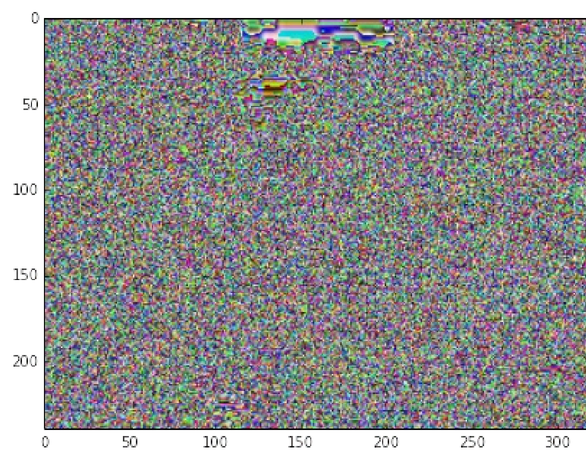


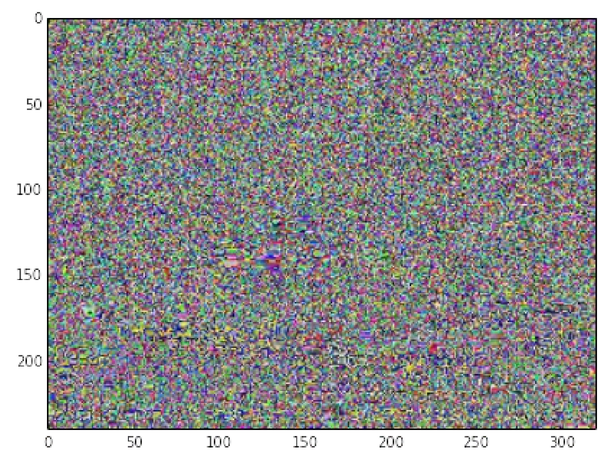


```
Test dataset
```


0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

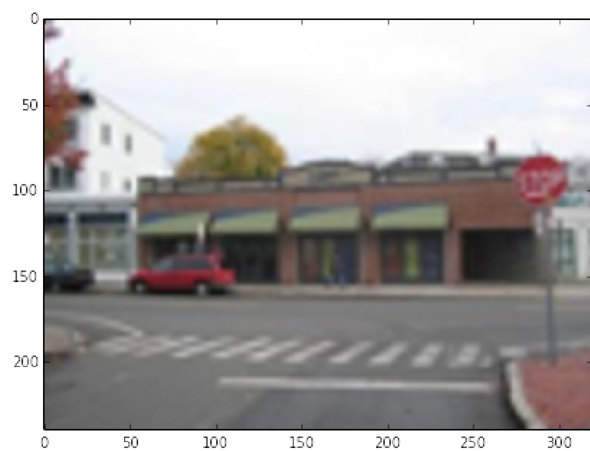

0
50
100
150
200
0
50
100
150
200
250
300

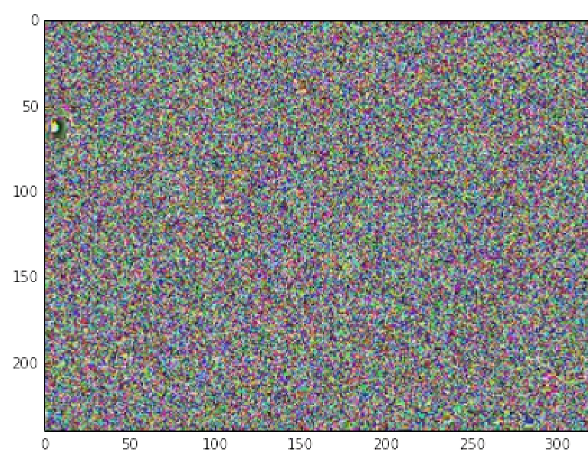

0
50
100
150
200
0
50
100
150
200
250
300

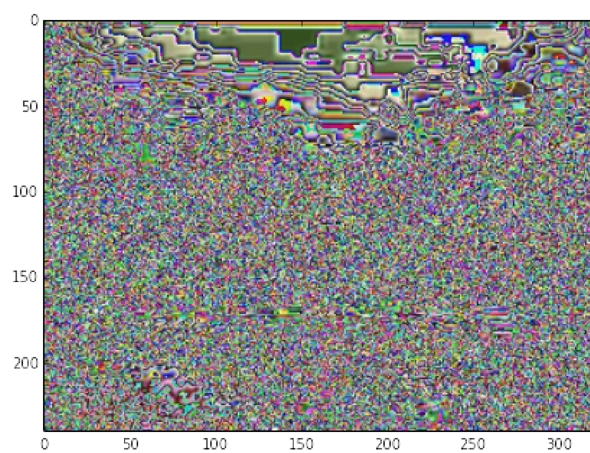

0
50
100
150
200
0
50
100
150
200
250
300

```
[1110/100000] cost: 349.5590
[1120/100000] cost: 268.0766
[1130/100000] cost: 302.1344
[1140/100000] cost: 208.0664
[1150/100000] cost: 264.6624
[1160/100000] cost: 222.6232
[1170/100000] cost: 272.3559
[1180/100000] cost: 233.2416
[1190/100000] cost: 231.7469
[1200/100000] cost: 236.6369
Training dataset
```









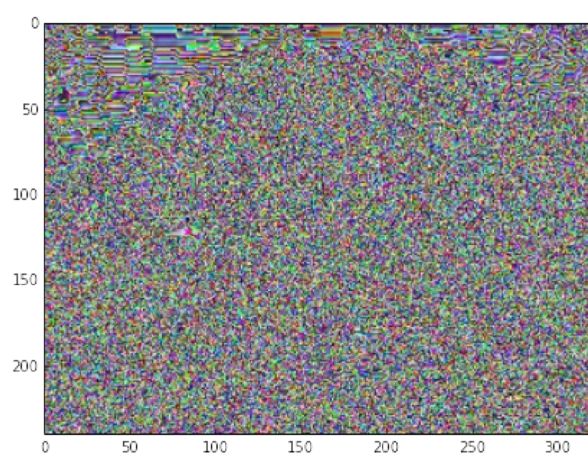


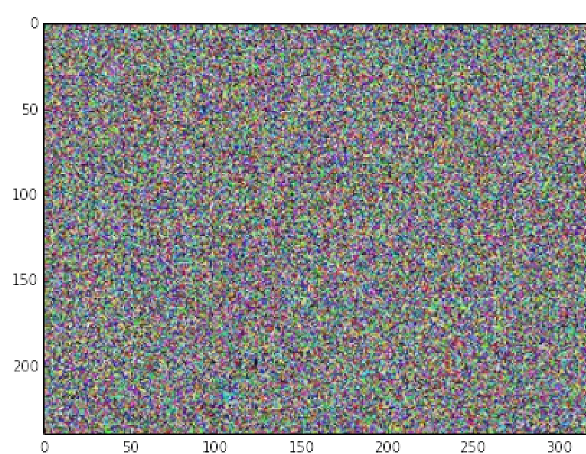



```
Test dataset
```


7
Würenlos
0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

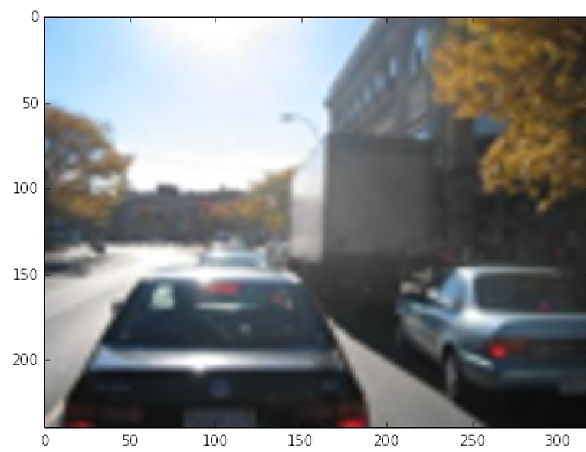
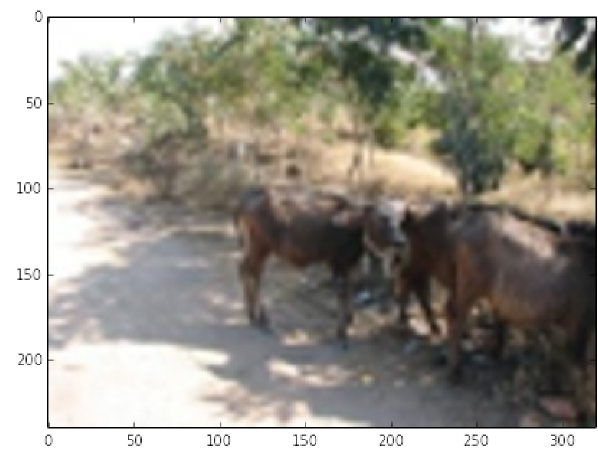
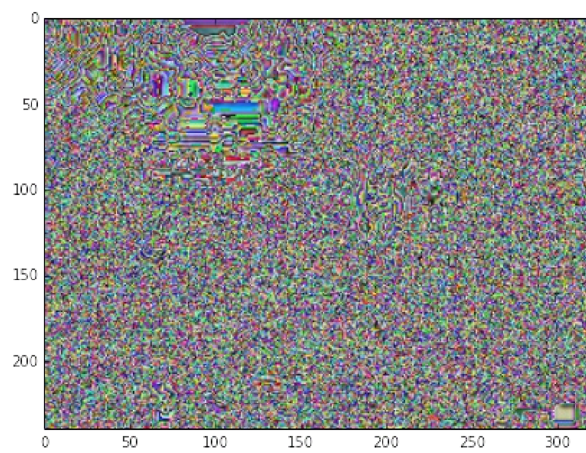
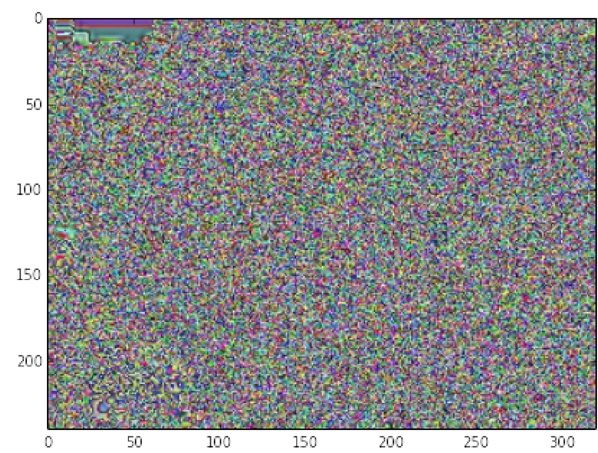
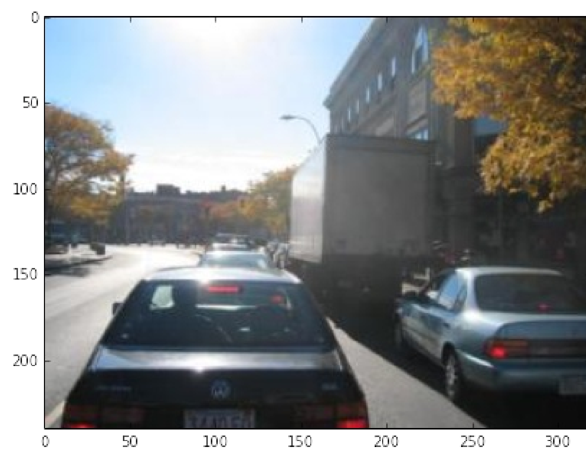
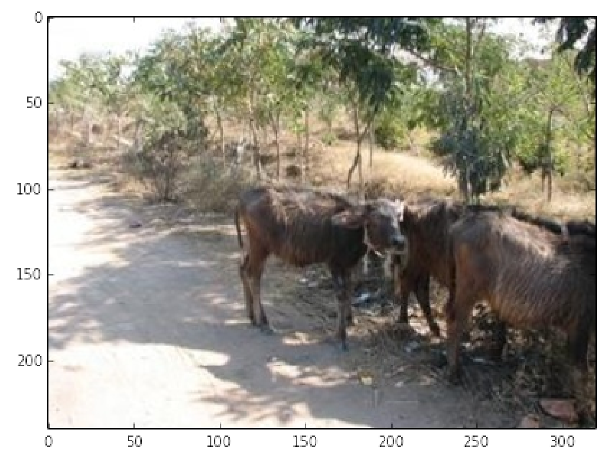

0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

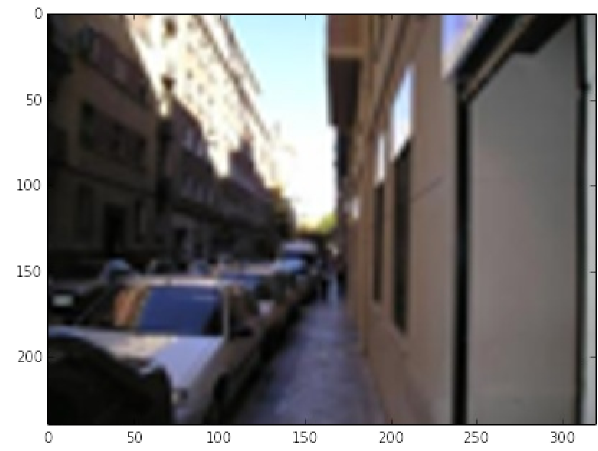
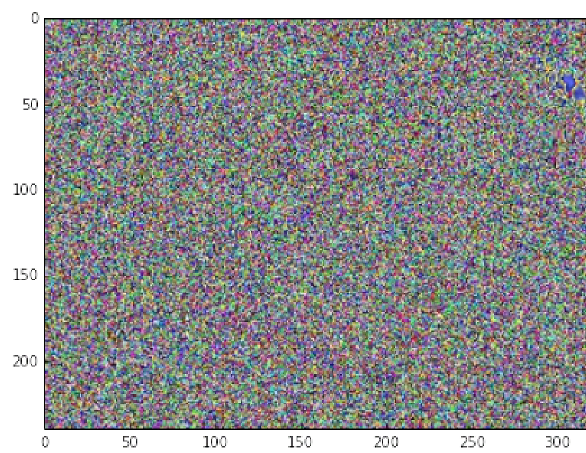
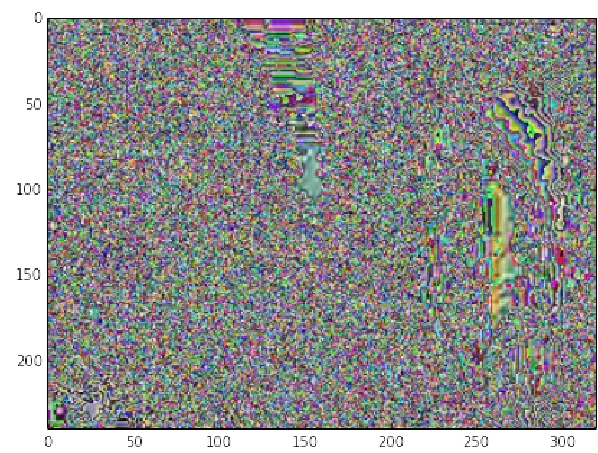

7
Würenlos
MAN
0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

```
[1210/100000] cost: 243.6459
[1220/100000] cost: 202.2522
[1230/100000] cost: 251.7291
[1240/100000] cost: 246.2129
[1250/100000] cost: 259.5334
[1260/100000] cost: 272.7007
[1270/100000] cost: 299.9875
[1280/100000] cost: 260.2490
[1290/100000] cost: 291.4149
[1300/100000] cost: 278.5886
Training dataset
```

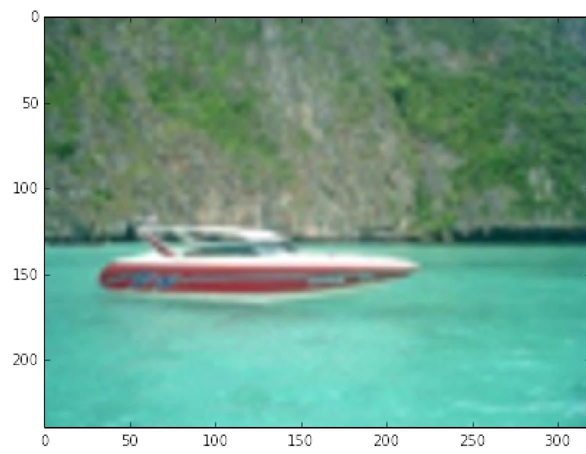







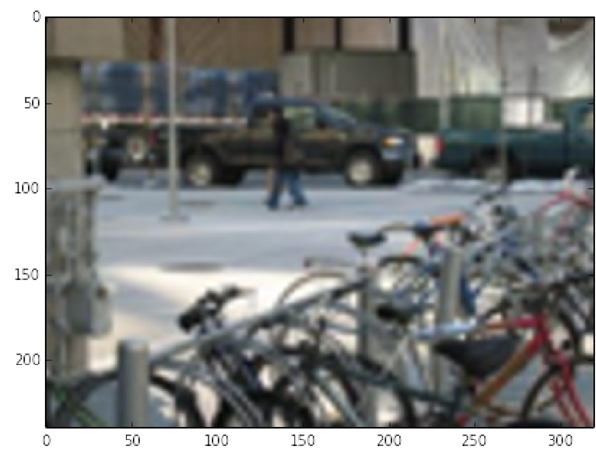
```
Test dataset
```

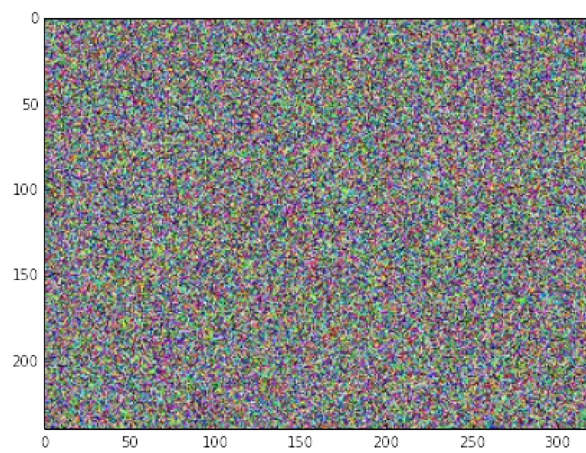





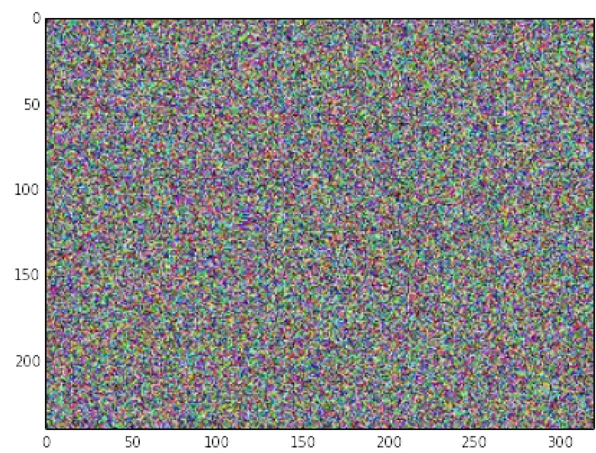


```
[1310/100000] cost: 260.8284
[1320/100000] cost: 246.8349
[1330/100000] cost: 355.7622
[1340/100000] cost: 287.9314
[1350/100000] cost: 253.2013
[1360/100000] cost: 294.0295
[1370/100000] cost: 249.6499
[1380/100000] cost: 270.4777
[1390/100000] cost: 253.6567
[1400/100000] cost: 258.3598
Training dataset
```





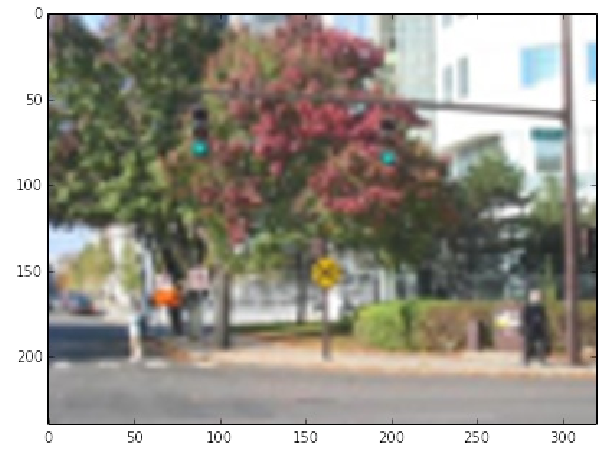
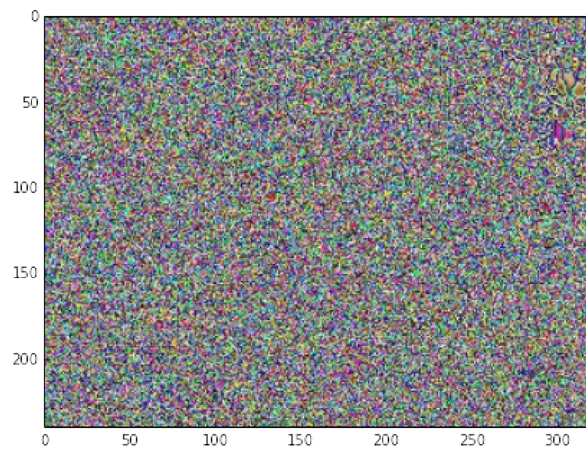
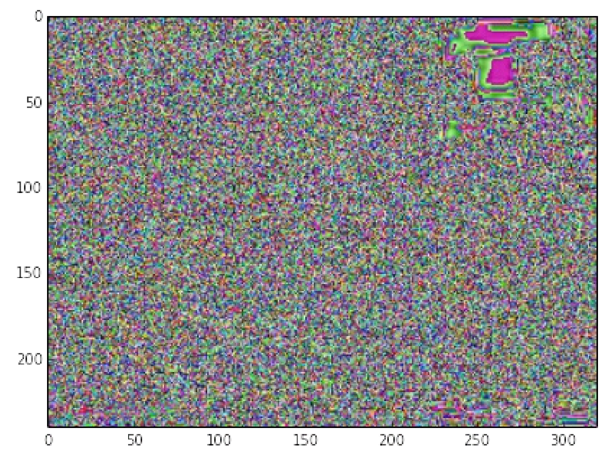








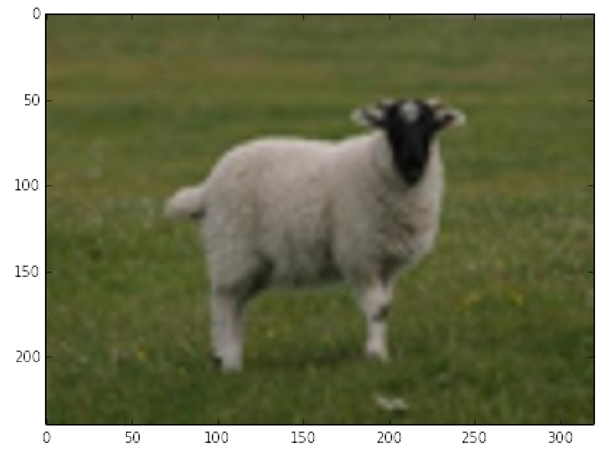
```
Test dataset
```

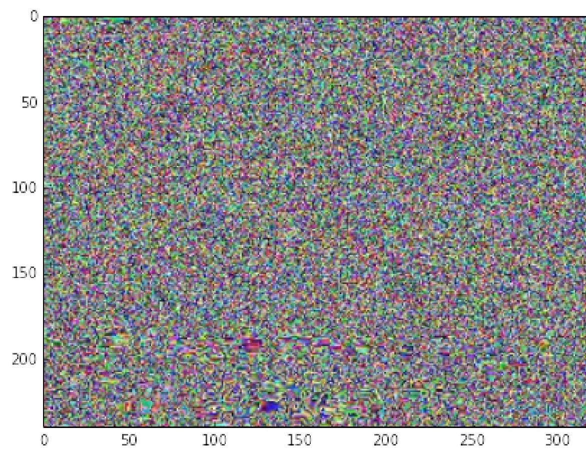







```
[1410/100000] cost: 252.8865
[1420/100000] cost: 226.4185
[1430/100000] cost: 274.6122
[1440/100000] cost: 284.6050
[1450/100000] cost: 275.3493
[1460/100000] cost: 286.1541
[1470/100000] cost: 300.7496
[1480/100000] cost: 258.0696
[1490/100000] cost: 302.6696
[1500/100000] cost: 321.3855
Training dataset
```

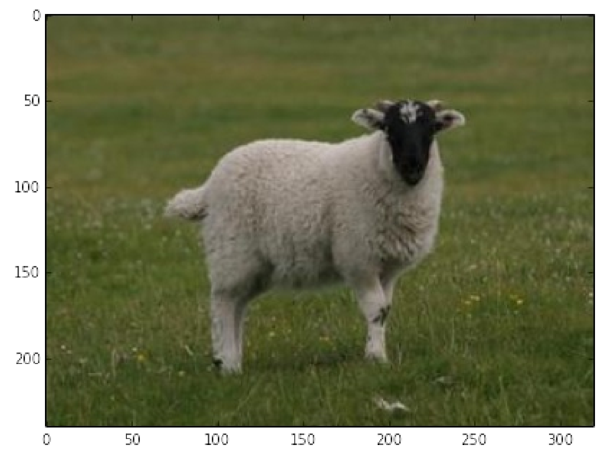




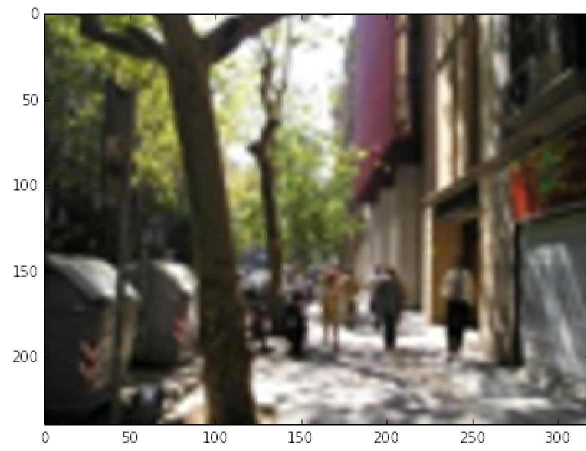




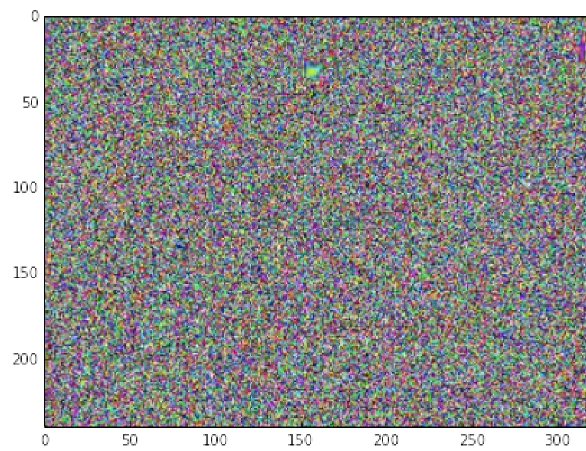





```
Test dataset
```





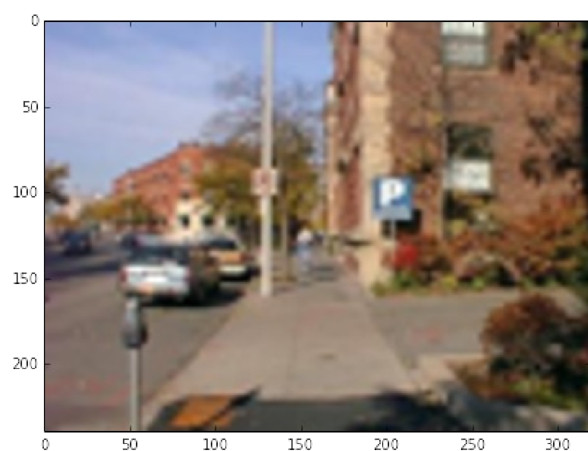




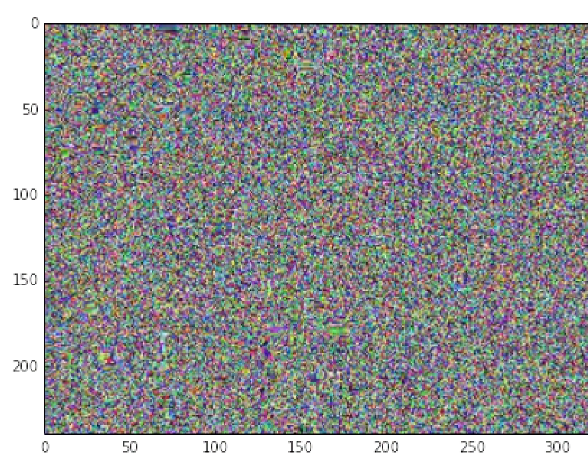


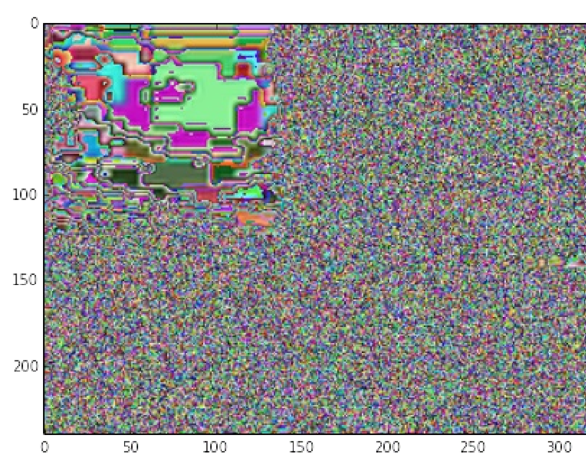


```
[1510/100000] cost: 268.9408
[1520/100000] cost: 305.9114
[1530/100000] cost: 301.6736
[1540/100000] cost: 267.8618
[1550/100000] cost: 323.6226
[1560/100000] cost: 265.2032
[1570/100000] cost: 318.1612
[1580/100000] cost: 256.5219
[1590/100000] cost: 300.4944
[1600/100000] cost: 305.1118
Training dataset
```





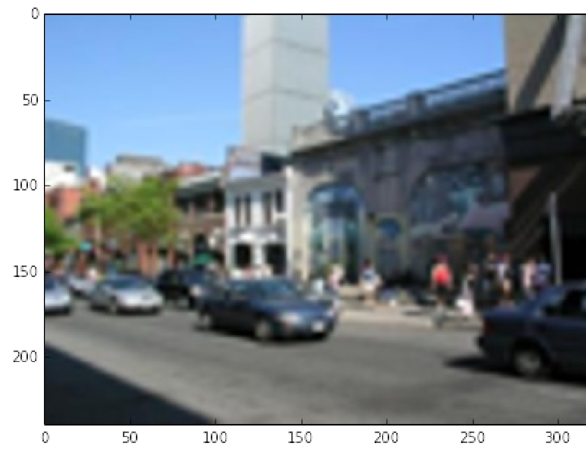
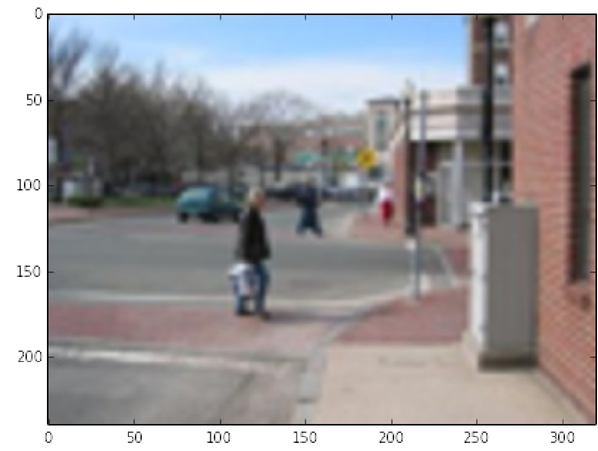
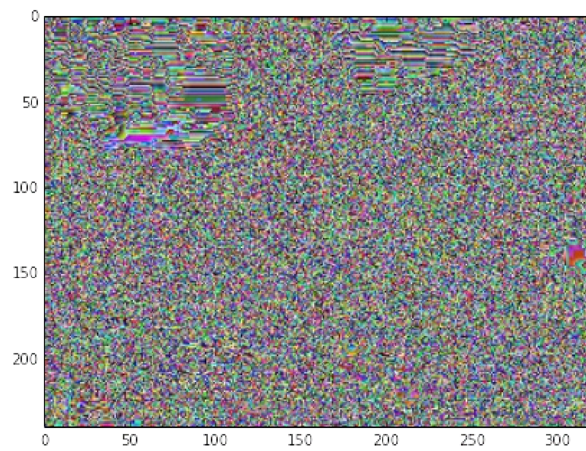
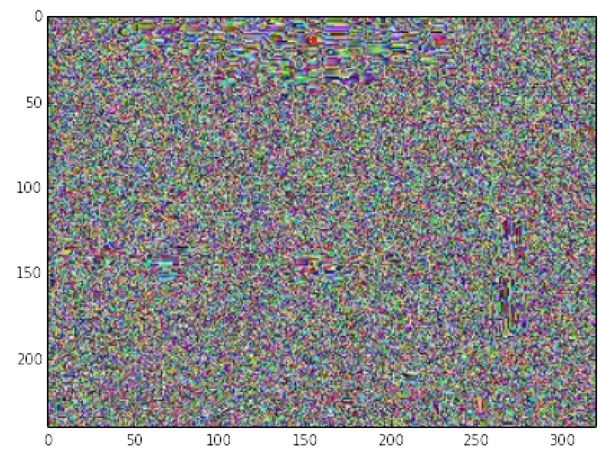
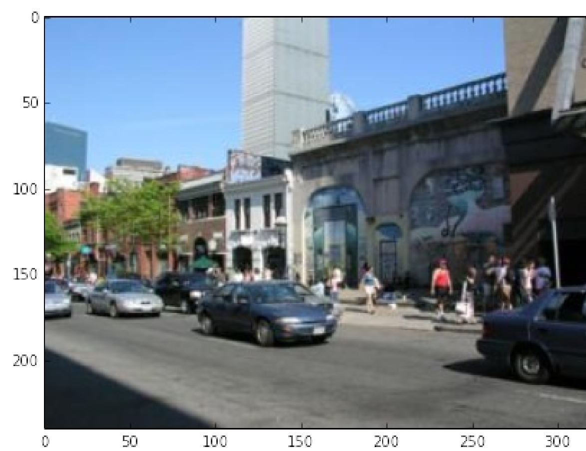
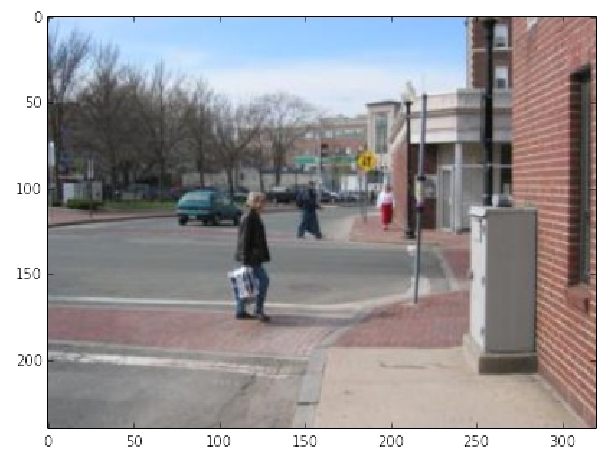




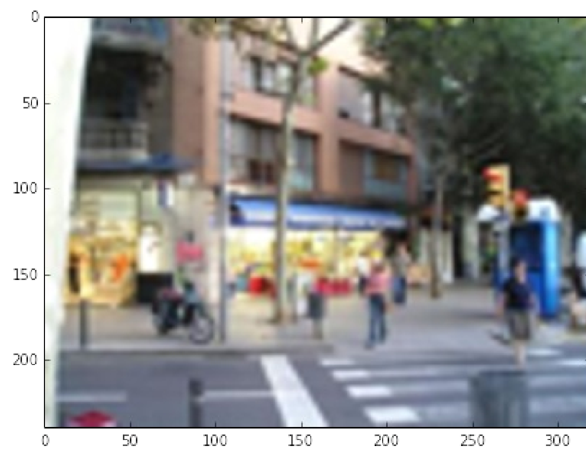


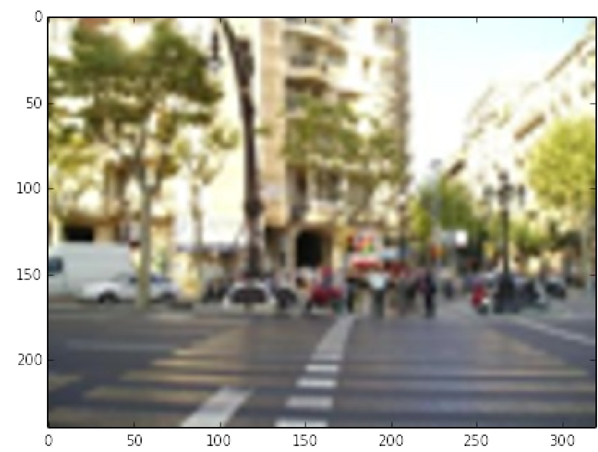


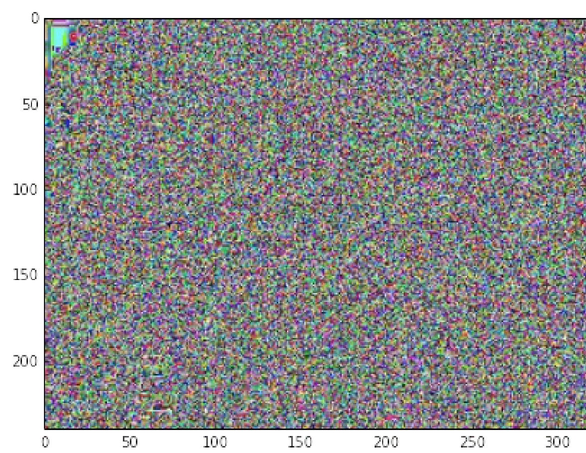

```
Test dataset
```

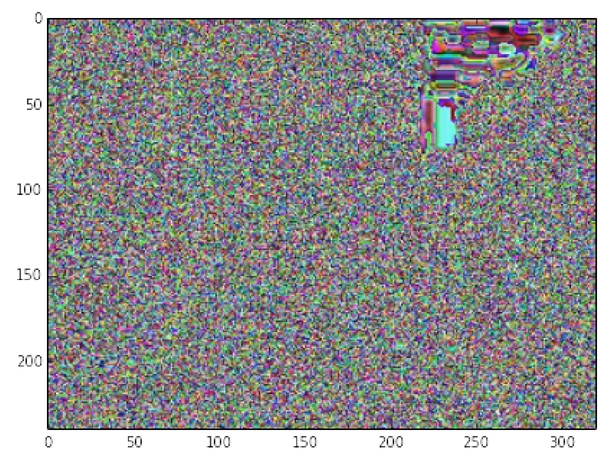







```
[1610/100000] cost: 272.6064
[1620/100000] cost: 284.0279
[1630/100000] cost: 317.0608
[1640/100000] cost: 278.9991
[1650/100000] cost: 305.5137
[1660/100000] cost: 271.6007
[1670/100000] cost: 287.9285
[1680/100000] cost: 299.2344
[1690/100000] cost: 284.0301
[1700/100000] cost: 248.4221
Training dataset
```





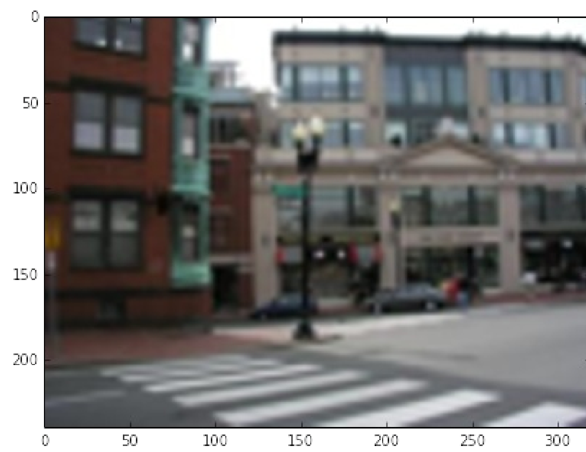
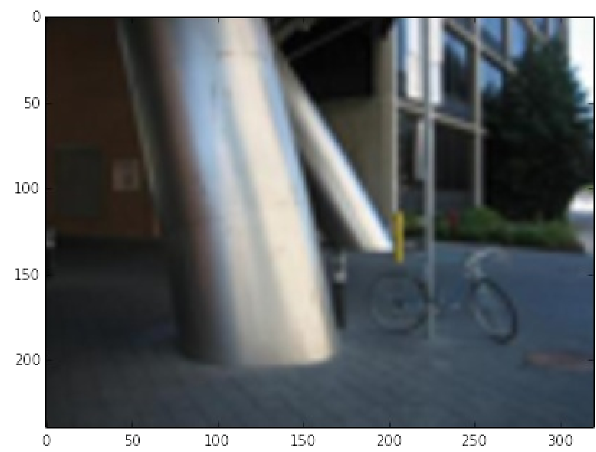
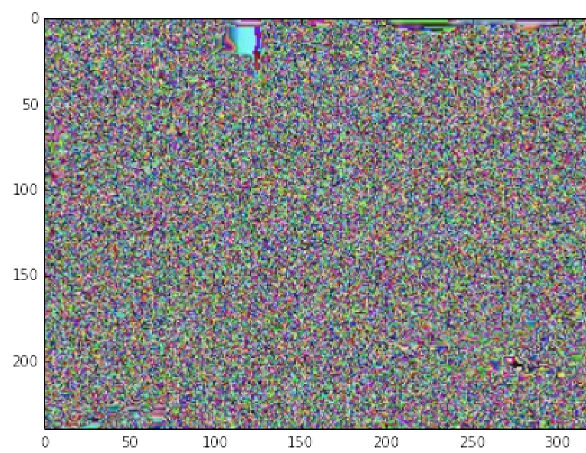
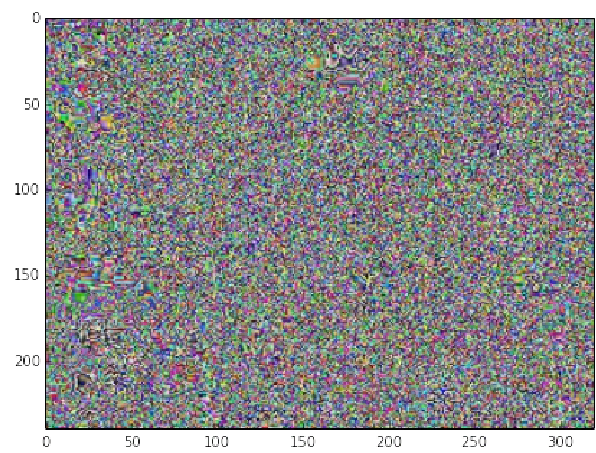


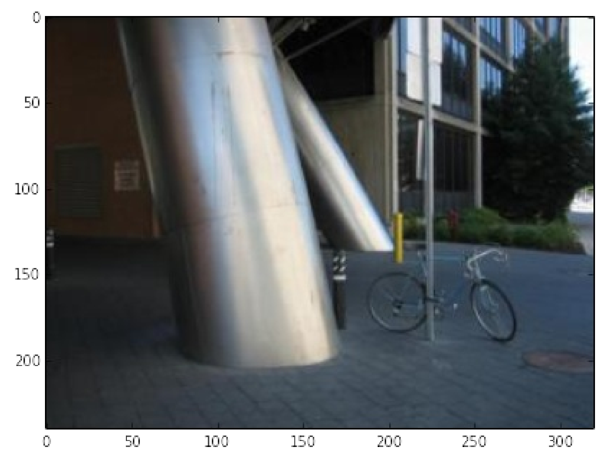







```
Test dataset
```



```
[1710/100000] cost: 280.1856
[1720/100000] cost: 337.9356
[1730/100000] cost: 267.5448
[1740/100000] cost: 334.8813
[1750/100000] cost: 302.4286
[1760/100000] cost: 324.7914
[1770/100000] cost: 258.5010
[1780/100000] cost: 327.7728
```

```
---------------------------------------------------------------------------
-----------

KeyboardInterrupt                         Traceback (most recent
 call last)

<ipython-input-8-b93cfefc0574> in <module>()
     11         batch_ys = ytrain[randidx, :]
     12         sess.run(optm, feed_dict={x: batch_xs
---> 13             , y: batch_ys, keepprob: 0.7})
     14     if (epoch_i % 10) == 0:
     15         print ("[%02d/%02d] cost: %.4f" % (epoch_i, n_ep
ochs


/usr/local/lib/python2.7/dist-packages/tensorflow/python/client/
session.pyc in run(self, fetches, feed_dict, options, run_metada
ta)
    338     try:
    339       result = self._run(None, fetches, feed_dict, optio
ns_ptr,
--> 340                          run_metadata_ptr)
    341       if run_metadata:
    342         proto_data = tf_session.TF_GetBuffer(run_metadat
a_ptr)


/usr/local/lib/python2.7/dist-packages/tensorflow/python/client/
session.pyc in _run(self, handle, fetches, feed_dict, options, r
un_metadata)
    562     try:
    563       results = self._do_run(handle, target_list, unique
_fetches,
--> 564                              feed_dict_string, options,
run_metadata)
    565     finally:
    566       # The movers are no longer used. Delete them.


/usr/local/lib/python2.7/dist-packages/tensorflow/python/client/
session.pyc in _do_run(self, handle, target_list, fetch_list, fe
ed_dict, options, run_metadata)
    635     if handle is None:
    636       return self._do_call(_run_fn, self._session, feed_
dict, fetch_list,
--> 637                            target_list, options, run_met
adata)
    638     else:
    639       return self._do_call(_prun_fn, self._session, hand
le, feed_dict,
```

```
/usr/local/lib/python2.7/dist-packages/tensorflow/python/client/
session.pyc in _do_call(self, fn, *args)
    642   def _do_call(self, fn, *args):
    643     try:
--> 644       return fn(*args)
    645     except tf_session.StatusNotOK as e:
    646       error_message = compat.as_text(e.error_message)


/usr/local/lib/python2.7/dist-packages/tensorflow/python/client/
session.pyc in _run_fn(session, feed_dict, fetch_list, target_li
st, options, run_metadata)
    626       else:
    627         return tf_session.TF_Run(
--> 628             session, None, feed_dict, fetch_list, target
_list, None)
    629
    630     def _prun_fn(session, handle, feed_dict, fetch_list)
:


KeyboardInterrupt:
```

```
import urllib
from bs4 import BeautifulSoup
from datetime import datetime
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline

print ("Packages loaded")
```

```
Packages loaded
```

Get it!

```
dates = []
date_strs = []
last_trade_prices = []
for nrpage in range(200):
    rawurl = "http://finance.naver.com/sise/sise_index_day.nhn?c
ode=KOSPI&page="
    url = rawurl + str(nrpage+1)
    # Get data (parse)
    soup = BeautifulSoup(urllib.urlopen(url).read())
    dateinfo  = soup.find_all('td', {'class': 'date'})
    valueinfo = soup.find_all('td', {'class': 'number_1'})
    nrdata    = len(valueinfo)
    for i in range(nrdata):
        # Date
        currdate = str(dateinfo[int(i/4)])
        currdate = currdate.replace('<td class="date">', '')
        currdate = currdate.replace('</td>', '')
        currdate = currdate.replace('.', "")

        # Values
        currdata = str(valueinfo[i])
        currdata = currdata.replace('<td class="number_1">', "")
        currdata = currdata.replace('</td>', "")
        currdata = currdata.replace('<span class="tah p11 red01"
>', '')
        currdata = currdata.replace('</span>', "")
        currdata = currdata.replace('<td class="number_1" style=
"padding-right:40px;">', "")
        currdata = currdata.replace('<td class="number_1" style=
"padding-right:30px;">', "")
        currdata = currdata.replace('<span class="tah p11 nv01">'
, "")
        currdata = currdata.replace(' ', "")
        currdata = currdata.replace('\n', "")
```

```
        currdata = currdata.replace('\t', "")
        currdata = currdata.replace(',', "")

        if i % 4 == 0:
            print ("\nCurr date is %s" % (currdate))
            date_strs.append(currdate)
            dates.append(float(currdate))
            print ("Last traded price: %s" % (currdata))
            last_trade_prices.append(float(currdata))
        elif i % 4 == 1:
            print ("Fluctuation ratio: %s" % (currdata))
        elif i % 4 == 2:
            print ("Traded volume:     %s" % (currdata))
        elif i % 4 == 3:
            print ("Traded price:      %s" % (currdata))
# Reverse
dates.reverse()
date_strs.reverse()
last_trade_prices.reverse()
```

```

Curr date is 20160725
Last traded price: 2009.52
Fluctuation ratio: -0.04%
Traded volume:     209464
Traded price:      2075669

Curr date is 20160722
Last traded price: 2010.34
Fluctuation ratio: -0.09%
Traded volume:     316365
Traded price:      3686869

Curr date is 20160721
Last traded price: 2012.22
Fluctuation ratio: -0.16%
Traded volume:     387481
Traded price:      3847464

Curr date is 20160720
Last traded price: 2015.46
Fluctuation ratio: -0.07%
Traded volume:     309989
Traded price:      3438452

Curr date is 20160719
Last traded price: 2016.89
Fluctuation ratio: -0.21%
Traded volume:     377425
Traded price:      3693836
```

```
Curr date is 20160718
Last traded price: 2021.11
Fluctuation ratio: +0.19%
Traded volume:     376256
Traded price:      3489650

Curr date is 20160715
Last traded price: 2017.26
Fluctuation ratio: +0.42%
Traded volume:     473600
Traded price:      4557938

Curr date is 20160714
Last traded price: 2008.77
Fluctuation ratio: +0.16%
Traded volume:     334351
Traded price:      4208693

Curr date is 20160713
Last traded price: 2005.55
Fluctuation ratio: +0.72%
Traded volume:     353365
Traded price:      4655778

Curr date is 20160712
Last traded price: 1991.23
Fluctuation ratio: +0.14%
Traded volume:     427454
Traded price:      4404322

Curr date is 20160711
Last traded price: 1988.54
Fluctuation ratio: +1.30%
Traded volume:     332552
Traded price:      4239228

Curr date is 20160708
Last traded price: 1963.10
Fluctuation ratio: -0.56%
Traded volume:     345926
Traded price:      4066529

Curr date is 20160707
Last traded price: 1974.08
Fluctuation ratio: +1.07%
Traded volume:     374574
Traded price:      3866686

Curr date is 20160706
Last traded price: 1953.12
Fluctuation ratio: -1.85%
Traded volume:     451690
Traded price:      4683910
```

```
Curr date is 20160705
Last traded price: 1989.85
Fluctuation ratio: -0.27%
Traded volume:     398730
Traded price:      3872453

Curr date is 20160704
Last traded price: 1995.30
Fluctuation ratio: +0.40%
Traded volume:     462342
Traded price:      4047771

Curr date is 20160701
Last traded price: 1987.32
Fluctuation ratio: +0.86%
Traded volume:     442076
Traded price:      4280276

Curr date is 20160630
Last traded price: 1970.35
Fluctuation ratio: +0.72%
Traded volume:     349862
Traded price:      5326567

Curr date is 20160629
Last traded price: 1956.36
Fluctuation ratio: +1.04%
Traded volume:     446686
Traded price:      4757342

Curr date is 20160628
Last traded price: 1936.22
Fluctuation ratio: +0.49%
Traded volume:     368239
Traded price:      4360447

Curr date is 20160627
Last traded price: 1926.85
Fluctuation ratio: +0.08%
Traded volume:     389617
Traded price:      5095563

Curr date is 20160624
Last traded price: 1925.24
Fluctuation ratio: -3.09%
Traded volume:     726661
Traded price:      8541520

Curr date is 20160623
Last traded price: 1986.71
Fluctuation ratio: -0.29%
Traded volume:     449586
```

```
Traded price:      4155747

Curr date is 20160622
Last traded price: 1992.58
Fluctuation ratio: +0.50%
Traded volume:     401930
Traded price:      3945907

Curr date is 20160621
Last traded price: 1982.70
Fluctuation ratio: +0.08%
Traded volume:     548571
Traded price:      4426426

Curr date is 20160620
Last traded price: 1981.12
Fluctuation ratio: +1.42%
Traded volume:     385489
Traded price:      3908118

Curr date is 20160617
Last traded price: 1953.40
Fluctuation ratio: +0.07%
Traded volume:     478811
Traded price:      4420744

Curr date is 20160616
Last traded price: 1951.99
Fluctuation ratio: -0.86%
Traded volume:     432990
Traded price:      4386606

Curr date is 20160615
Last traded price: 1968.83
Fluctuation ratio: -0.16%
Traded volume:     371503
Traded price:      4491931

Curr date is 20160614
Last traded price: 1972.03
Fluctuation ratio: -0.36%
Traded volume:     434390
Traded price:      4919384

Curr date is 20160613
Last traded price: 1979.06
Fluctuation ratio: -1.91%
Traded volume:     437223
Traded price:      4530726

Curr date is 20160610
Last traded price: 2017.63
Fluctuation ratio: -0.32%
```

```
Traded volume:     399304
Traded price:      4711178

Curr date is 20160609
Last traded price: 2024.17
Fluctuation ratio: -0.14%
Traded volume:     591484
Traded price:      7585446

Curr date is 20160608
Last traded price: 2027.08
Fluctuation ratio: +0.77%
Traded volume:     490074
Traded price:      5705455

Curr date is 20160607
Last traded price: 2011.63
Fluctuation ratio: +1.30%
Traded volume:     472781
Traded price:      5459279

Curr date is 20160603
Last traded price: 1985.84
Fluctuation ratio: +0.04%
Traded volume:     514910
Traded price:      5150261

Curr date is 20160602
Last traded price: 1985.11
Fluctuation ratio: +0.12%
Traded volume:     461076
Traded price:      5143907

Curr date is 20160601
Last traded price: 1982.72
Fluctuation ratio: -0.03%
Traded volume:     496335
Traded price:      5314023

Curr date is 20160531
Last traded price: 1983.40
Fluctuation ratio: +0.83%
Traded volume:     647543
Traded price:      8425308

Curr date is 20160530
Last traded price: 1967.13
Fluctuation ratio: -0.10%
Traded volume:     541202
Traded price:      4532734

Curr date is 20160527
Last traded price: 1969.17
```

```
Fluctuation ratio: +0.62%
Traded volume:     517989
Traded price:      4814077

Curr date is 20160526
Last traded price: 1957.06
Fluctuation ratio: -0.18%
Traded volume:     551033
Traded price:      4991688

Curr date is 20160525
Last traded price: 1960.51
Fluctuation ratio: +1.18%
Traded volume:     650749
Traded price:      5410461

Curr date is 20160524
Last traded price: 1937.68
Fluctuation ratio: -0.90%
Traded volume:     511809
Traded price:      4905045

Curr date is 20160523
Last traded price: 1955.25
Fluctuation ratio: +0.39%
Traded volume:     393845
Traded price:      4011176

Curr date is 20160520
Last traded price: 1947.67
Fluctuation ratio: +0.05%
Traded volume:     383159
Traded price:      4399090

Curr date is 20160519
Last traded price: 1946.78
Fluctuation ratio: -0.51%
Traded volume:     500755
Traded price:      5225527

Curr date is 20160518
Last traded price: 1956.73
Fluctuation ratio: -0.58%
Traded volume:     688631
Traded price:      6511596

Curr date is 20160517
Last traded price: 1968.06
Fluctuation ratio: +0.01%
Traded volume:     590808
Traded price:      5414916

Curr date is 20160516
```

```
Last traded price: 1967.91
Fluctuation ratio: +0.05%
Traded volume:     520295
Traded price:      4797446

Curr date is 20160513
Last traded price: 1966.99
Fluctuation ratio: -0.53%
Traded volume:     500776
Traded price:      4495599

Curr date is 20160512
Last traded price: 1977.49
Fluctuation ratio: -0.13%
Traded volume:     445986
Traded price:      4038195

Curr date is 20160511
Last traded price: 1980.10
Fluctuation ratio: -0.12%
Traded volume:     393495
Traded price:      4363815

Curr date is 20160510
Last traded price: 1982.50
Fluctuation ratio: +0.75%
Traded volume:     364980
Traded price:      4773150

Curr date is 20160509
Last traded price: 1967.81
Fluctuation ratio: -0.45%
Traded volume:     338214
Traded price:      5179317

Curr date is 20160504
Last traded price: 1976.71
Fluctuation ratio: -0.49%
Traded volume:     348731
Traded price:      4926327

Curr date is 20160503
Last traded price: 1986.41
Fluctuation ratio: +0.42%
Traded volume:     306496
Traded price:      3729473

Curr date is 20160502
Last traded price: 1978.15
Fluctuation ratio: -0.80%
Traded volume:     324975
Traded price:      3849348
```

```
Curr date is 20160429
Last traded price: 1994.15
Fluctuation ratio: -0.34%
Traded volume:     393295
Traded price:      4899464

Curr date is 20160428
Last traded price: 2000.93
Fluctuation ratio: -0.72%
Traded volume:     336073
Traded price:      5240945

Curr date is 20160427
Last traded price: 2015.40
Fluctuation ratio: -0.21%
Traded volume:     407874
Traded price:      5312008

Curr date is 20160426
Last traded price: 2019.63
Fluctuation ratio: +0.25%
Traded volume:     502391
Traded price:      5110466

Curr date is 20160425
Last traded price: 2014.55
Fluctuation ratio: -0.05%
Traded volume:     392813
Traded price:      4310191

Curr date is 20160422
Last traded price: 2015.49
Fluctuation ratio: -0.33%
Traded volume:     475058
Traded price:      4149096

Curr date is 20160421
Last traded price: 2022.10
Fluctuation ratio: +0.81%
Traded volume:     387844
Traded price:      4714212

Curr date is 20160420
Last traded price: 2005.83
Fluctuation ratio: -0.27%
Traded volume:     419041
Traded price:      4528325

Curr date is 20160419
Last traded price: 2011.36
Fluctuation ratio: +0.11%
Traded volume:     397413
Traded price:      4392297
```

```
Curr date is 20160418
Last traded price: 2009.10
Fluctuation ratio: -0.28%
Traded volume:     334757
Traded price:      4247470

Curr date is 20160415
Last traded price: 2014.71
Fluctuation ratio: -0.06%
Traded volume:     313110
Traded price:      4428494

Curr date is 20160414
Last traded price: 2015.93
Fluctuation ratio: +1.75%
Traded volume:     456829
Traded price:      6285943

Curr date is 20160412
Last traded price: 1981.32
Fluctuation ratio: +0.56%
Traded volume:     372146
Traded price:      3861385

Curr date is 20160411
Last traded price: 1970.37
Fluctuation ratio: -0.09%
Traded volume:     370220
Traded price:      3801200

Curr date is 20160408
Last traded price: 1972.05
Fluctuation ratio: -0.09%
Traded volume:     394571
Traded price:      4332858

Curr date is 20160407
Last traded price: 1973.89
Fluctuation ratio: +0.13%
Traded volume:     427063
Traded price:      4578002

Curr date is 20160406
Last traded price: 1971.32
Fluctuation ratio: +0.44%
Traded volume:     318532
Traded price:      4299791

Curr date is 20160405
Last traded price: 1962.74
Fluctuation ratio: -0.82%
Traded volume:     359996
```

```
Traded price:      3992300

Curr date is 20160404
Last traded price: 1978.97
Fluctuation ratio: +0.27%
Traded volume:     359061
Traded price:      4192246

Curr date is 20160401
Last traded price: 1973.57
Fluctuation ratio: -1.12%
Traded volume:     378202
Traded price:      4777024

Curr date is 20160331
Last traded price: 1995.85
Fluctuation ratio: -0.31%
Traded volume:     322879
Traded price:      4840902

Curr date is 20160330
Last traded price: 2002.14
Fluctuation ratio: +0.36%
Traded volume:     347263
Traded price:      4517708

Curr date is 20160329
Last traded price: 1994.91
Fluctuation ratio: +0.62%
Traded volume:     377908
Traded price:      3991750

Curr date is 20160328
Last traded price: 1982.54
Fluctuation ratio: -0.06%
Traded volume:     405980
Traded price:      3365429

Curr date is 20160325
Last traded price: 1983.81
Fluctuation ratio: -0.11%
Traded volume:     457306
Traded price:      3751912

Curr date is 20160324
Last traded price: 1985.97
Fluctuation ratio: -0.46%
Traded volume:     336185
Traded price:      3777427

Curr date is 20160323
Last traded price: 1995.12
Fluctuation ratio: -0.08%
```

```
Traded volume:     363163
Traded price:      3929377

Curr date is 20160322
Last traded price: 1996.81
Fluctuation ratio: +0.35%
Traded volume:     379381
Traded price:      4416765

Curr date is 20160321
Last traded price: 1989.76
Fluctuation ratio: -0.12%
Traded volume:     320666
Traded price:      4012257

Curr date is 20160318
Last traded price: 1992.12
Fluctuation ratio: +0.21%
Traded volume:     380108
Traded price:      4659018

Curr date is 20160317
Last traded price: 1987.99
Fluctuation ratio: +0.66%
Traded volume:     282237
Traded price:      4668378

Curr date is 20160316
Last traded price: 1974.90
Fluctuation ratio: +0.25%
Traded volume:     289945
Traded price:      4055964

Curr date is 20160315
Last traded price: 1969.97
Fluctuation ratio: -0.12%
Traded volume:     260916
Traded price:      3764672

Curr date is 20160314
Last traded price: 1972.27
Fluctuation ratio: +0.04%
Traded volume:     353027
Traded price:      4167785

Curr date is 20160311
Last traded price: 1971.41
Fluctuation ratio: +0.11%
Traded volume:     299293
Traded price:      4116595

Curr date is 20160310
Last traded price: 1969.33
```

```
Fluctuation ratio: +0.84%
Traded volume:     346406
Traded price:      5055951

Curr date is 20160309
Last traded price: 1952.95
Fluctuation ratio: +0.35%
Traded volume:     345920
Traded price:      4212533

Curr date is 20160308
Last traded price: 1946.12
Fluctuation ratio: -0.60%
Traded volume:     354037
Traded price:      4844001

Curr date is 20160307
Last traded price: 1957.87
Fluctuation ratio: +0.11%
Traded volume:     340260
Traded price:      4611978

Curr date is 20160304
Last traded price: 1955.63
Fluctuation ratio: -0.13%
Traded volume:     312175
Traded price:      4491198

Curr date is 20160303
Last traded price: 1958.17
Fluctuation ratio: +0.55%
Traded volume:     417546
Traded price:      4903735

Curr date is 20160302
Last traded price: 1947.42
Fluctuation ratio: +1.60%
Traded volume:     316865
Traded price:      5110434

Curr date is 20160229
Last traded price: 1916.66
Fluctuation ratio: -0.18%
Traded volume:     271547
Traded price:      4534806

Curr date is 20160226
Last traded price: 1920.16
Fluctuation ratio: +0.08%
Traded volume:     268811
Traded price:      3964062

Curr date is 20160225
```

```
Last traded price: 1918.57
Fluctuation ratio: +0.32%
Traded volume:     297760
Traded price:      3881831

Curr date is 20160224
Last traded price: 1912.53
Fluctuation ratio: -0.09%
Traded volume:     277072
Traded price:      3516031

Curr date is 20160223
Last traded price: 1914.22
Fluctuation ratio: -0.11%
Traded volume:     323952
Traded price:      3948979

Curr date is 20160222
Last traded price: 1916.36
Fluctuation ratio: +0.01%
Traded volume:     261987
Traded price:      3404998

Curr date is 20160219
Last traded price: 1916.24
Fluctuation ratio: +0.39%
Traded volume:     307221
Traded price:      3825333

Curr date is 20160218
Last traded price: 1908.84
Fluctuation ratio: +1.32%
Traded volume:     327418
Traded price:      4646452

Curr date is 20160217
Last traded price: 1883.94
Fluctuation ratio: -0.23%
Traded volume:     333899
Traded price:      5061065

Curr date is 20160216
Last traded price: 1888.30
Fluctuation ratio: +1.40%
Traded volume:     381620
Traded price:      4196969

Curr date is 20160215
Last traded price: 1862.20
Fluctuation ratio: +1.47%
Traded volume:     281431
Traded price:      4413358
```

```
Curr date is 20160212
Last traded price: 1835.28
Fluctuation ratio: -1.41%
Traded volume:     465667
Traded price:      6236294

Curr date is 20160211
Last traded price: 1861.54
Fluctuation ratio: -2.93%
Traded volume:     381071
Traded price:      4635376

Curr date is 20160205
Last traded price: 1917.79
Fluctuation ratio: +0.08%
Traded volume:     290560
Traded price:      4245650

Curr date is 20160204
Last traded price: 1916.26
Fluctuation ratio: +1.35%
Traded volume:     358802
Traded price:      5131794

Curr date is 20160203
Last traded price: 1890.67
Fluctuation ratio: -0.84%
Traded volume:     403016
Traded price:      4739686

Curr date is 20160202
Last traded price: 1906.60
Fluctuation ratio: -0.95%
Traded volume:     407067
Traded price:      4552972

Curr date is 20160201
Last traded price: 1924.82
Fluctuation ratio: +0.67%
Traded volume:     435759
Traded price:      5062947

Curr date is 20160129
Last traded price: 1912.06
Fluctuation ratio: +0.27%
Traded volume:     379159
Traded price:      5966488

Curr date is 20160128
Last traded price: 1906.94
Fluctuation ratio: +0.48%
Traded volume:     306801
Traded price:      4609354
```

```
Curr date is 20160127
Last traded price: 1897.87
Fluctuation ratio: +1.40%
Traded volume:     345542
Traded price:      4999182

Curr date is 20160126
Last traded price: 1871.69
Fluctuation ratio: -1.15%
Traded volume:     295266
Traded price:      4409881

Curr date is 20160125
Last traded price: 1893.43
Fluctuation ratio: +0.74%
Traded volume:     311413
Traded price:      4004975

Curr date is 20160122
Last traded price: 1879.43
Fluctuation ratio: +2.11%
Traded volume:     400293
Traded price:      4344515

Curr date is 20160121
Last traded price: 1840.53
Fluctuation ratio: -0.27%
Traded volume:     424485
Traded price:      4729614

Curr date is 20160120
Last traded price: 1845.45
Fluctuation ratio: -2.34%
Traded volume:     400261
Traded price:      5240327

Curr date is 20160119
Last traded price: 1889.64
Fluctuation ratio: +0.60%
Traded volume:     329977
Traded price:      4441550

Curr date is 20160118
Last traded price: 1878.45
Fluctuation ratio: -0.02%
Traded volume:     273261
Traded price:      4056222

Curr date is 20160115
Last traded price: 1878.87
Fluctuation ratio: -1.11%
Traded volume:     303991
```

```
Traded price:      4391132

Curr date is 20160114
Last traded price: 1900.01
Fluctuation ratio: -0.85%
Traded volume:     305931
Traded price:      4264454

Curr date is 20160113
Last traded price: 1916.28
Fluctuation ratio: +1.34%
Traded volume:     261810
Traded price:      3882822

Curr date is 20160112
Last traded price: 1890.86
Fluctuation ratio: -0.21%
Traded volume:     290808
Traded price:      3787415

Curr date is 20160111
Last traded price: 1894.84
Fluctuation ratio: -1.19%
Traded volume:     319556
Traded price:      4145526

Curr date is 20160108
Last traded price: 1917.62
Fluctuation ratio: +0.70%
Traded volume:     400556
Traded price:      4995379

Curr date is 20160107
Last traded price: 1904.33
Fluctuation ratio: -1.10%
Traded volume:     388136
Traded price:      4907247

Curr date is 20160106
Last traded price: 1925.43
Fluctuation ratio: -0.26%
Traded volume:     585978
Traded price:      5917977

Curr date is 20160105
Last traded price: 1930.53
Fluctuation ratio: +0.61%
Traded volume:     440914
Traded price:      4097962

Curr date is 20160104
Last traded price: 1918.76
Fluctuation ratio: -2.17%
```

```
Traded volume:     355001
Traded price:      3917340

Curr date is 20151230
Last traded price: 1961.31
Fluctuation ratio: -0.25%
Traded volume:     300574
Traded price:      3670486

Curr date is 20151229
Last traded price: 1966.31
Fluctuation ratio: +0.11%
Traded volume:     395493
Traded price:      3757607

Curr date is 20151228
Last traded price: 1964.06
Fluctuation ratio: -1.34%
Traded volume:     389692
Traded price:      3753973

Curr date is 20151224
Last traded price: 1990.65
Fluctuation ratio: -0.43%
Traded volume:     497593
Traded price:      3551535

Curr date is 20151223
Last traded price: 1999.22
Fluctuation ratio: +0.33%
Traded volume:     487032
Traded price:      3918294

Curr date is 20151222
Last traded price: 1992.56
Fluctuation ratio: +0.57%
Traded volume:     569183
Traded price:      4119722

Curr date is 20151221
Last traded price: 1981.19
Fluctuation ratio: +0.30%
Traded volume:     532535
Traded price:      3534909

Curr date is 20151218
Last traded price: 1975.32
Fluctuation ratio: -0.13%
Traded volume:     665731
Traded price:      4021859

Curr date is 20151217
Last traded price: 1977.96
```

```
Fluctuation ratio: +0.43%
Traded volume:     374572
Traded price:      3850731

Curr date is 20151216
Last traded price: 1969.40
Fluctuation ratio: +1.88%
Traded volume:     467637
Traded price:      4207020

Curr date is 20151215
Last traded price: 1932.97
Fluctuation ratio: +0.27%
Traded volume:     427891
Traded price:      3593520

Curr date is 20151214
Last traded price: 1927.82
Fluctuation ratio: -1.07%
Traded volume:     425668
Traded price:      4022027

Curr date is 20151211
Last traded price: 1948.62
Fluctuation ratio: -0.18%
Traded volume:     373906
Traded price:      4020271

Curr date is 20151210
Last traded price: 1952.07
Fluctuation ratio: +0.20%
Traded volume:     372100
Traded price:      4506365

Curr date is 20151209
Last traded price: 1948.24
Fluctuation ratio: -0.04%
Traded volume:     373794
Traded price:      3636724

Curr date is 20151208
Last traded price: 1949.04
Fluctuation ratio: -0.75%
Traded volume:     401162
Traded price:      3574227

Curr date is 20151207
Last traded price: 1963.67
Fluctuation ratio: -0.54%
Traded volume:     376919
Traded price:      3356172

Curr date is 20151204
```

```
Last traded price: 1974.40
Fluctuation ratio: -0.99%
Traded volume:     373376
Traded price:      3174983

Curr date is 20151203
Last traded price: 1994.07
Fluctuation ratio: -0.76%
Traded volume:     360920
Traded price:      3434293

Curr date is 20151202
Last traded price: 2009.29
Fluctuation ratio: -0.72%
Traded volume:     623964
Traded price:      4341114

Curr date is 20151201
Last traded price: 2023.93
Fluctuation ratio: +1.60%
Traded volume:     565138
Traded price:      4672220

Curr date is 20151130
Last traded price: 1991.97
Fluctuation ratio: -1.82%
Traded volume:     446112
Traded price:      6409955

Curr date is 20151127
Last traded price: 2028.99
Fluctuation ratio: -0.08%
Traded volume:     475264
Traded price:      3898119

Curr date is 20151126
Last traded price: 2030.68
Fluctuation ratio: +1.06%
Traded volume:     447540
Traded price:      4361336

Curr date is 20151125
Last traded price: 2009.42
Fluctuation ratio: -0.34%
Traded volume:     387046
Traded price:      4170392

Curr date is 20151124
Last traded price: 2016.29
Fluctuation ratio: +0.63%
Traded volume:     409942
Traded price:      4228723
```

```
Curr date is 20151123
Last traded price: 2003.70
Fluctuation ratio: +0.70%
Traded volume:     379028
Traded price:      4422655

Curr date is 20151120
Last traded price: 1989.86
Fluctuation ratio: +0.05%
Traded volume:     409589
Traded price:      3981101

Curr date is 20151119
Last traded price: 1988.91
Fluctuation ratio: +1.33%
Traded volume:     402673
Traded price:      4469284

Curr date is 20151118
Last traded price: 1962.88
Fluctuation ratio: -0.04%
Traded volume:     533967
Traded price:      4539628

Curr date is 20151117
Last traded price: 1963.58
Fluctuation ratio: +1.06%
Traded volume:     411426
Traded price:      4414119

Curr date is 20151116
Last traded price: 1943.02
Fluctuation ratio: -1.53%
Traded volume:     421007
Traded price:      4762680

Curr date is 20151113
Last traded price: 1973.29
Fluctuation ratio: -1.01%
Traded volume:     337727
Traded price:      4186209

Curr date is 20151112
Last traded price: 1993.36
Fluctuation ratio: -0.20%
Traded volume:     382858
Traded price:      3402325

Curr date is 20151111
Last traded price: 1997.27
Fluctuation ratio: +0.03%
Traded volume:     282596
Traded price:      3795769
```

```
Curr date is 20151110
Last traded price: 1996.59
Fluctuation ratio: -1.44%
Traded volume:     409528
Traded price:      5372521

Curr date is 20151109
Last traded price: 2025.70
Fluctuation ratio: -0.75%
Traded volume:     430279
Traded price:      5932869

Curr date is 20151106
Last traded price: 2041.07
Fluctuation ratio: -0.41%
Traded volume:     458898
Traded price:      4513121

Curr date is 20151105
Last traded price: 2049.41
Fluctuation ratio: -0.16%
Traded volume:     344157
Traded price:      3946581

Curr date is 20151104
Last traded price: 2052.77
Fluctuation ratio: +0.21%
Traded volume:     531883
Traded price:      4854033

Curr date is 20151103
Last traded price: 2048.40
Fluctuation ratio: +0.65%
Traded volume:     582904
Traded price:      5731116

Curr date is 20151102
Last traded price: 2035.24
Fluctuation ratio: +0.28%
Traded volume:     357975
Traded price:      4670602

Curr date is 20151030
Last traded price: 2029.47
Fluctuation ratio: -0.23%
Traded volume:     373689
Traded price:      5945879

Curr date is 20151029
Last traded price: 2034.16
Fluctuation ratio: -0.41%
Traded volume:     409167
```

```
Traded price:      5622179

Curr date is 20151028
Last traded price: 2042.51
Fluctuation ratio: -0.10%
Traded volume:     387254
Traded price:      4877015

Curr date is 20151027
Last traded price: 2044.65
Fluctuation ratio: -0.17%
Traded volume:     441782
Traded price:      4011414

Curr date is 20151026
Last traded price: 2048.08
Fluctuation ratio: +0.38%
Traded volume:     455015
Traded price:      4267516

Curr date is 20151023
Last traded price: 2040.40
Fluctuation ratio: +0.86%
Traded volume:     641858
Traded price:      5303923

Curr date is 20151022
Last traded price: 2023.00
Fluctuation ratio: -0.98%
Traded volume:     688427
Traded price:      5877354

Curr date is 20151021
Last traded price: 2042.98
Fluctuation ratio: +0.18%
Traded volume:     660815
Traded price:      5306063

Curr date is 20151020
Last traded price: 2039.36
Fluctuation ratio: +0.45%
Traded volume:     945413
Traded price:      4691672

Curr date is 20151019
Last traded price: 2030.27
Fluctuation ratio: +0.00%
Traded volume:     707700
Traded price:      4210325

Curr date is 20151016
Last traded price: 2030.26
Fluctuation ratio: -0.15%
```

```
Traded volume:     542027
Traded price:      4616085

Curr date is 20151015
Last traded price: 2033.27
Fluctuation ratio: +1.18%
Traded volume:     489792
Traded price:      4564600

Curr date is 20151014
Last traded price: 2009.55
Fluctuation ratio: -0.47%
Traded volume:     634915
Traded price:      4959508

Curr date is 20151013
Last traded price: 2019.05
Fluctuation ratio: -0.13%
Traded volume:     595222
Traded price:      5112232

Curr date is 20151012
Last traded price: 2021.63
Fluctuation ratio: +0.10%
Traded volume:     624399
Traded price:      5988383

Curr date is 20151008
Last traded price: 2019.53
Fluctuation ratio: +0.68%
Traded volume:     693705
Traded price:      6067223

Curr date is 20151007
Last traded price: 2005.84
Fluctuation ratio: +0.76%
Traded volume:     753341
Traded price:      6090499

Curr date is 20151006
Last traded price: 1990.65
Fluctuation ratio: +0.63%
Traded volume:     653014
Traded price:      5457343

Curr date is 20151005
Last traded price: 1978.25
Fluctuation ratio: +0.44%
Traded volume:     539542
Traded price:      4533462

Curr date is 20151002
Last traded price: 1969.68
```

```
Fluctuation ratio: -0.49%
Traded volume:     1020136
Traded price:      5192845

Curr date is 20151001
Last traded price: 1979.32
Fluctuation ratio: +0.84%
Traded volume:     670916
Traded price:      5093572

Curr date is 20150930
Last traded price: 1962.81
Fluctuation ratio: +1.03%
Traded volume:     539384
Traded price:      5435749

Curr date is 20150925
Last traded price: 1942.85
Fluctuation ratio: -0.22%
Traded volume:     789819
Traded price:      4176784

Curr date is 20150924
Last traded price: 1947.10
Fluctuation ratio: +0.13%
Traded volume:     895574
Traded price:      4373185

Curr date is 20150923
Last traded price: 1944.64
Fluctuation ratio: -1.89%
Traded volume:     845584
Traded price:      4853314

Curr date is 20150922
Last traded price: 1982.06
Fluctuation ratio: +0.88%
Traded volume:     683342
Traded price:      4456248

Curr date is 20150921
Last traded price: 1964.68
Fluctuation ratio: -1.57%
Traded volume:     372130
Traded price:      3755076

Curr date is 20150918
Last traded price: 1995.95
Fluctuation ratio: +0.98%
Traded volume:     515916
Traded price:      6663946

Curr date is 20150917
```

```
Last traded price: 1976.49
Fluctuation ratio: +0.05%
Traded volume:     502903
Traded price:      5545797

Curr date is 20150916
Last traded price: 1975.45
Fluctuation ratio: +1.96%
Traded volume:     577385
Traded price:      5836409

Curr date is 20150915
Last traded price: 1937.56
Fluctuation ratio: +0.32%
Traded volume:     386186
Traded price:      3928182

Curr date is 20150914
Last traded price: 1931.46
Fluctuation ratio: -0.51%
Traded volume:     359481
Traded price:      4086021

Curr date is 20150911
Last traded price: 1941.37
Fluctuation ratio: -1.06%
Traded volume:     497246
Traded price:      4605410

Curr date is 20150910
Last traded price: 1962.11
Fluctuation ratio: +1.44%
Traded volume:     455738
Traded price:      5396708

Curr date is 20150909
Last traded price: 1934.20
Fluctuation ratio: +2.96%
Traded volume:     508956
Traded price:      5193258

Curr date is 20150908
Last traded price: 1878.68
Fluctuation ratio: -0.24%
Traded volume:     386234
Traded price:      4676248

Curr date is 20150907
Last traded price: 1883.22
Fluctuation ratio: -0.15%
Traded volume:     285330
Traded price:      3730988
```

```
Curr date is 20150904
Last traded price: 1886.04
Fluctuation ratio: -1.54%
Traded volume:     357577
Traded price:      4642584

Curr date is 20150903
Last traded price: 1915.53
Fluctuation ratio: +0.02%
Traded volume:     293126
Traded price:      4513384

Curr date is 20150902
Last traded price: 1915.22
Fluctuation ratio: +0.05%
Traded volume:     338567
Traded price:      4953189

Curr date is 20150901
Last traded price: 1914.23
Fluctuation ratio: -1.40%
Traded volume:     320859
Traded price:      4782631

Curr date is 20150831
Last traded price: 1941.49
Fluctuation ratio: +0.20%
Traded volume:     346953
Traded price:      5950573

Curr date is 20150828
Last traded price: 1937.67
Fluctuation ratio: +1.56%
Traded volume:     439192
Traded price:      6405456

Curr date is 20150827
Last traded price: 1908.00
Fluctuation ratio: +0.73%
Traded volume:     477548
Traded price:      6640239

Curr date is 20150826
Last traded price: 1894.09
Fluctuation ratio: +2.57%
Traded volume:     448166
Traded price:      6763249

Curr date is 20150825
Last traded price: 1846.63
Fluctuation ratio: +0.92%
Traded volume:     446401
Traded price:      6564908
```

```
Curr date is 20150824
Last traded price: 1829.81
Fluctuation ratio: -2.47%
Traded volume:     410169
Traded price:      6088766

Curr date is 20150821
Last traded price: 1876.07
Fluctuation ratio: -2.01%
Traded volume:     506532
Traded price:      6761186

Curr date is 20150820
Last traded price: 1914.55
Fluctuation ratio: -1.28%
Traded volume:     389843
Traded price:      5077380

Curr date is 20150819
Last traded price: 1939.38
Fluctuation ratio: -0.86%
Traded volume:     498925
Traded price:      7107639

Curr date is 20150818
Last traded price: 1956.26
Fluctuation ratio: -0.62%
Traded volume:     468191
Traded price:      5533960

Curr date is 20150817
Last traded price: 1968.52
Fluctuation ratio: -0.75%
Traded volume:     556074
Traded price:      4561509

Curr date is 20150813
Last traded price: 1983.46
Fluctuation ratio: +0.40%
Traded volume:     373204
Traded price:      5114020

Curr date is 20150812
Last traded price: 1975.47
Fluctuation ratio: -0.56%
Traded volume:     519701
Traded price:      6801476

Curr date is 20150811
Last traded price: 1986.65
Fluctuation ratio: -0.82%
Traded volume:     362930
```

```
Traded price:       5029071

Curr date is 20150810
Last traded price: 2003.17
Fluctuation ratio: -0.35%
Traded volume:     305473
Traded price:       4450795

Curr date is 20150807
Last traded price: 2010.23
Fluctuation ratio: -0.15%
Traded volume:     330275
Traded price:       4538493

Curr date is 20150806
Last traded price: 2013.29
Fluctuation ratio: -0.81%
Traded volume:     335637
Traded price:       5738406

Curr date is 20150805
Last traded price: 2029.76
Fluctuation ratio: +0.09%
Traded volume:     314253
Traded price:       4977507

Curr date is 20150804
Last traded price: 2027.99
Fluctuation ratio: +0.97%
Traded volume:     384528
Traded price:       4864611

Curr date is 20150803
Last traded price: 2008.49
Fluctuation ratio: -1.07%
Traded volume:     351899
Traded price:       4969419

Curr date is 20150731
Last traded price: 2030.16
Fluctuation ratio: +0.55%
Traded volume:     444009
Traded price:       6187274

Curr date is 20150730
Last traded price: 2019.03
Fluctuation ratio: -0.91%
Traded volume:     503234
Traded price:       6690278

Curr date is 20150729
Last traded price: 2037.62
Fluctuation ratio: -0.07%
```

```
Traded volume:     514386
Traded price:      6590690

Curr date is 20150728
Last traded price: 2039.10
Fluctuation ratio: +0.01%
Traded volume:     490449
Traded price:      6440757

Curr date is 20150727
Last traded price: 2038.81
Fluctuation ratio: -0.35%
Traded volume:     535705
Traded price:      6235263

Curr date is 20150724
Last traded price: 2045.96
Fluctuation ratio: -0.93%
Traded volume:     604456
Traded price:      6021543

Curr date is 20150723
Last traded price: 2065.07
Fluctuation ratio: +0.02%
Traded volume:     732798
Traded price:      6137884

Curr date is 20150722
Last traded price: 2064.73
Fluctuation ratio: -0.91%
Traded volume:     1149355
Traded price:      6766264

Curr date is 20150721
Last traded price: 2083.62
Fluctuation ratio: +0.50%
Traded volume:     718786
Traded price:      6488347

Curr date is 20150720
Last traded price: 2073.31
Fluctuation ratio: -0.17%
Traded volume:     862846
Traded price:      5968491

Curr date is 20150717
Last traded price: 2076.79
Fluctuation ratio: -0.53%
Traded volume:     806118
Traded price:      8319283

Curr date is 20150716
Last traded price: 2087.89
```

```
Fluctuation ratio: +0.72%
Traded volume:     682603
Traded price:      7059502

Curr date is 20150715
Last traded price: 2072.91
Fluctuation ratio: +0.66%
Traded volume:     613189
Traded price:      7492762

Curr date is 20150714
Last traded price: 2059.23
Fluctuation ratio: -0.11%
Traded volume:     496984
Traded price:      6687802

Curr date is 20150713
Last traded price: 2061.52
Fluctuation ratio: +1.49%
Traded volume:     421534
Traded price:      6183981

Curr date is 20150710
Last traded price: 2031.17
Fluctuation ratio: +0.17%
Traded volume:     371847
Traded price:      6514152

Curr date is 20150709
Last traded price: 2027.81
Fluctuation ratio: +0.58%
Traded volume:     403722
Traded price:      6994503

Curr date is 20150708
Last traded price: 2016.21
Fluctuation ratio: -1.18%
Traded volume:     447415
Traded price:      7392498

Curr date is 20150707
Last traded price: 2040.29
Fluctuation ratio: -0.66%
Traded volume:     497988
Traded price:      8200258

Curr date is 20150706
Last traded price: 2053.93
Fluctuation ratio: -2.40%
Traded volume:     463773
Traded price:      6275913

Curr date is 20150703
```

```
Last traded price: 2104.41
Fluctuation ratio: -0.14%
Traded volume:     414027
Traded price:      5877241

Curr date is 20150702
Last traded price: 2107.33
Fluctuation ratio: +0.45%
Traded volume:     559474
Traded price:      6954702

Curr date is 20150701
Last traded price: 2097.89
Fluctuation ratio: +1.14%
Traded volume:     468233
Traded price:      5804594

Curr date is 20150630
Last traded price: 2074.20
Fluctuation ratio: +0.67%
Traded volume:     389704
Traded price:      5606673

Curr date is 20150629
Last traded price: 2060.49
Fluctuation ratio: -1.42%
Traded volume:     499301
Traded price:      6142787

Curr date is 20150626
Last traded price: 2090.26
Fluctuation ratio: +0.25%
Traded volume:     487506
Traded price:      6232862

Curr date is 20150625
Last traded price: 2085.06
Fluctuation ratio: -0.02%
Traded volume:     569595
Traded price:      5832424

Curr date is 20150624
Last traded price: 2085.53
Fluctuation ratio: +0.21%
Traded volume:     394955
Traded price:      5240680

Curr date is 20150623
Last traded price: 2081.20
Fluctuation ratio: +1.27%
Traded volume:     325895
Traded price:      5218253
```

```
Curr date is 20150622
Last traded price: 2055.16
Fluctuation ratio: +0.40%
Traded volume:     262775
Traded price:      4248248

Curr date is 20150619
Last traded price: 2046.96
Fluctuation ratio: +0.25%
Traded volume:     394065
Traded price:      5570859

Curr date is 20150618
Last traded price: 2041.88
Fluctuation ratio: +0.34%
Traded volume:     386776
Traded price:      5245838

Curr date is 20150617
Last traded price: 2034.86
Fluctuation ratio: +0.30%
Traded volume:     349919
Traded price:      5064150

Curr date is 20150616
Last traded price: 2028.72
Fluctuation ratio: -0.67%
Traded volume:     376207
Traded price:      6265626

Curr date is 20150615
Last traded price: 2042.32
Fluctuation ratio: -0.48%
Traded volume:     310139
Traded price:      4765898

Curr date is 20150612
Last traded price: 2052.17
Fluctuation ratio: -0.22%
Traded volume:     384132
Traded price:      5960234

Curr date is 20150611
Last traded price: 2056.61
Fluctuation ratio: +0.26%
Traded volume:     451273
Traded price:      7495997

Curr date is 20150610
Last traded price: 2051.32
Fluctuation ratio: -0.62%
Traded volume:     418420
Traded price:      6101561
```

```
Curr date is 20150609
Last traded price: 2064.03
Fluctuation ratio: -0.06%
Traded volume:     418684
Traded price:      5538722

Curr date is 20150608
Last traded price: 2065.19
Fluctuation ratio: -0.14%
Traded volume:     390357
Traded price:      5508814

Curr date is 20150605
Last traded price: 2068.10
Fluctuation ratio: -0.23%
Traded volume:     388853
Traded price:      6086742

Curr date is 20150604
Last traded price: 2072.86
Fluctuation ratio: +0.47%
Traded volume:     392034
Traded price:      7004163

Curr date is 20150603
Last traded price: 2063.16
Fluctuation ratio: -0.74%
Traded volume:     532758
Traded price:      6300056

Curr date is 20150602
Last traded price: 2078.64
Fluctuation ratio: -1.13%
Traded volume:     515192
Traded price:      6940036

Curr date is 20150601
Last traded price: 2102.37
Fluctuation ratio: -0.59%
Traded volume:     420716
Traded price:      5697910

Curr date is 20150529
Last traded price: 2114.80
Fluctuation ratio: +0.19%
Traded volume:     448189
Traded price:      7648634

Curr date is 20150528
Last traded price: 2110.89
Fluctuation ratio: +0.16%
Traded volume:     410706
```

```
Traded price:      6742808

Curr date is 20150527
Last traded price: 2107.50
Fluctuation ratio: -1.68%
Traded volume:     434088
Traded price:      7887325

Curr date is 20150526
Last traded price: 2143.50
Fluctuation ratio: -0.12%
Traded volume:     373921
Traded price:      6372057

Curr date is 20150522
Last traded price: 2146.10
Fluctuation ratio: +1.10%
Traded volume:     436973
Traded price:      5992137

Curr date is 20150521
Last traded price: 2122.81
Fluctuation ratio: -0.78%
Traded volume:     392880
Traded price:      5596351

Curr date is 20150520
Last traded price: 2139.54
Fluctuation ratio: +0.88%
Traded volume:     447646
Traded price:      5998522

Curr date is 20150519
Last traded price: 2120.85
Fluctuation ratio: +0.34%
Traded volume:     411965
Traded price:      5650762

Curr date is 20150518
Last traded price: 2113.72
Fluctuation ratio: +0.34%
Traded volume:     441445
Traded price:      5784617

Curr date is 20150515
Last traded price: 2106.50
Fluctuation ratio: -0.65%
Traded volume:     467268
Traded price:      5365887

Curr date is 20150514
Last traded price: 2120.33
Fluctuation ratio: +0.29%
```

```
Traded volume:     425176
Traded price:      5003053

Curr date is 20150513
Last traded price: 2114.16
Fluctuation ratio: +0.83%
Traded volume:     373063
Traded price:      4928946

Curr date is 20150512
Last traded price: 2096.77
Fluctuation ratio: -0.03%
Traded volume:     386675
Traded price:      4899718

Curr date is 20150511
Last traded price: 2097.38
Fluctuation ratio: +0.57%
Traded volume:     326946
Traded price:      4962776

Curr date is 20150508
Last traded price: 2085.52
Fluctuation ratio: -0.26%
Traded volume:     305314
Traded price:      4677202

Curr date is 20150507
Last traded price: 2091.00
Fluctuation ratio: -0.65%
Traded volume:     418759
Traded price:      5370714

Curr date is 20150506
Last traded price: 2104.58
Fluctuation ratio: -1.30%
Traded volume:     424198
Traded price:      5600790

Curr date is 20150504
Last traded price: 2132.23
Fluctuation ratio: +0.24%
Traded volume:     352069
Traded price:      4668413

Curr date is 20150430
Last traded price: 2127.17
Fluctuation ratio: -0.72%
Traded volume:     497732
Traded price:      6018805

Curr date is 20150429
Last traded price: 2142.63
```

```
Fluctuation ratio: -0.23%
Traded volume:     453869
Traded price:      6564697

Curr date is 20150428
Last traded price: 2147.67
Fluctuation ratio: -0.46%
Traded volume:     410562
Traded price:      6068630

Curr date is 20150427
Last traded price: 2157.54
Fluctuation ratio: -0.10%
Traded volume:     388660
Traded price:      6117139

Curr date is 20150424
Last traded price: 2159.80
Fluctuation ratio: -0.63%
Traded volume:     503418
Traded price:      8114606

Curr date is 20150423
Last traded price: 2173.41
Fluctuation ratio: +1.38%
Traded volume:     514837
Traded price:      8076510

Curr date is 20150422
Last traded price: 2143.89
Fluctuation ratio: -0.04%
Traded volume:     762349
Traded price:      8388538

Curr date is 20150421
Last traded price: 2144.79
Fluctuation ratio: -0.09%
Traded volume:     555234
Traded price:      6825206

Curr date is 20150420
Last traded price: 2146.71
Fluctuation ratio: +0.15%
Traded volume:     503401
Traded price:      6973925

Curr date is 20150417
Last traded price: 2143.50
Fluctuation ratio: +0.17%
Traded volume:     457908
Traded price:      6249221

Curr date is 20150416
```

```
Last traded price: 2139.90
Fluctuation ratio: +0.94%
Traded volume:     471230
Traded price:      7187304

Curr date is 20150415
Last traded price: 2119.96
Fluctuation ratio: +0.39%
Traded volume:     445868
Traded price:      7283611

Curr date is 20150414
Last traded price: 2111.72
Fluctuation ratio: +0.61%
Traded volume:     636475
Traded price:      7954023

Curr date is 20150413
Last traded price: 2098.92
Fluctuation ratio: +0.53%
Traded volume:     550382
Traded price:      6949775

Curr date is 20150410
Last traded price: 2087.76
Fluctuation ratio: +1.40%
Traded volume:     647107
Traded price:      6275483

Curr date is 20150409
Last traded price: 2058.87
Fluctuation ratio: -0.02%
Traded volume:     519429
Traded price:      5805350

Curr date is 20150408
Last traded price: 2059.26
Fluctuation ratio: +0.60%
Traded volume:     546028
Traded price:      6416068

Curr date is 20150407
Last traded price: 2047.03
Fluctuation ratio: +0.03%
Traded volume:     426271
Traded price:      4760116

Curr date is 20150406
Last traded price: 2046.43
Fluctuation ratio: +0.05%
Traded volume:     454031
Traded price:      4726562
```

```
Curr date is 20150403
Last traded price: 2045.42
Fluctuation ratio: +0.81%
Traded volume:     493143
Traded price:      4452256

Curr date is 20150402
Last traded price: 2029.07
Fluctuation ratio: +0.03%
Traded volume:     646302
Traded price:      5904789

Curr date is 20150401
Last traded price: 2028.45
Fluctuation ratio: -0.62%
Traded volume:     538580
Traded price:      5316792

Curr date is 20150331
Last traded price: 2041.03
Fluctuation ratio: +0.54%
Traded volume:     547252
Traded price:      5291411

Curr date is 20150330
Last traded price: 2030.04
Fluctuation ratio: +0.51%
Traded volume:     435179
Traded price:      4347308

Curr date is 20150327
Last traded price: 2019.80
Fluctuation ratio: -0.14%
Traded volume:     442814
Traded price:      5035104

Curr date is 20150326
Last traded price: 2022.56
Fluctuation ratio: -0.99%
Traded volume:     338831
Traded price:      5100235

Curr date is 20150325
Last traded price: 2042.81
Fluctuation ratio: +0.07%
Traded volume:     418100
Traded price:      4983230

Curr date is 20150324
Last traded price: 2041.37
Fluctuation ratio: +0.23%
Traded volume:     324351
Traded price:      4561843
```

```
Curr date is 20150323
Last traded price: 2036.59
Fluctuation ratio: -0.03%
Traded volume:     310208
Traded price:      4575023

Curr date is 20150320
Last traded price: 2037.24
Fluctuation ratio: -0.03%
Traded volume:     347283
Traded price:      5060596

Curr date is 20150319
Last traded price: 2037.89
Fluctuation ratio: +0.47%
Traded volume:     445165
Traded price:      5557805

Curr date is 20150318
Last traded price: 2028.45
Fluctuation ratio: -0.07%
Traded volume:     358261
Traded price:      5306022

Curr date is 20150317
Last traded price: 2029.91
Fluctuation ratio: +2.14%
Traded volume:     353014
Traded price:      5331677

Curr date is 20150316
Last traded price: 1987.33
Fluctuation ratio: +0.08%
Traded volume:     315596
Traded price:      4562851

Curr date is 20150313
Last traded price: 1985.79
Fluctuation ratio: +0.77%
Traded volume:     322560
Traded price:      4249807

Curr date is 20150312
Last traded price: 1970.59
Fluctuation ratio: -0.52%
Traded volume:     379584
Traded price:      6277284

Curr date is 20150311
Last traded price: 1980.83
Fluctuation ratio: -0.20%
Traded volume:     395371
```

```
Traded price:      5795408

Curr date is 20150310
Last traded price: 1984.77
Fluctuation ratio: -0.40%
Traded volume:     430271
Traded price:      4677577

Curr date is 20150309
Last traded price: 1992.82
Fluctuation ratio: -1.00%
Traded volume:     292973
Traded price:      3941996

Curr date is 20150306
Last traded price: 2012.94
Fluctuation ratio: +0.73%
Traded volume:     389914
Traded price:      4749043

Curr date is 20150305
Last traded price: 1998.38
Fluctuation ratio: +0.00%
Traded volume:     362427
Traded price:      3939052

Curr date is 20150304
Last traded price: 1998.29
Fluctuation ratio: -0.15%
Traded volume:     369997
Traded price:      4595754

Curr date is 20150303
Last traded price: 2001.38
Fluctuation ratio: +0.23%
Traded volume:     440885
Traded price:      5301006

Curr date is 20150302
Last traded price: 1996.81
Fluctuation ratio: +0.55%
Traded volume:     365957
Traded price:      5493257

Curr date is 20150227
Last traded price: 1985.80
Fluctuation ratio: -0.37%
Traded volume:     417184
Traded price:      5527749

Curr date is 20150226
Last traded price: 1993.08
Fluctuation ratio: +0.13%
```

```
Traded volume:     379351
Traded price:      5019255

Curr date is 20150225
Last traded price: 1990.47
Fluctuation ratio: +0.73%
Traded volume:     360343
Traded price:      4827387

Curr date is 20150224
Last traded price: 1976.12
Fluctuation ratio: +0.39%
Traded volume:     304394
Traded price:      4112593

Curr date is 20150223
Last traded price: 1968.39
Fluctuation ratio: +0.35%
Traded volume:     334135
Traded price:      4038723

Curr date is 20150217
Last traded price: 1961.45
Fluctuation ratio: +0.16%
Traded volume:     298178
Traded price:      3461146

Curr date is 20150216
Last traded price: 1958.23
Fluctuation ratio: +0.04%
Traded volume:     344836
Traded price:      3637341

Curr date is 20150213
Last traded price: 1957.50
Fluctuation ratio: +0.82%
Traded volume:     342798
Traded price:      4011442

Curr date is 20150212
Last traded price: 1941.63
Fluctuation ratio: -0.21%
Traded volume:     327123
Traded price:      4263274

Curr date is 20150211
Last traded price: 1945.70
Fluctuation ratio: +0.51%
Traded volume:     300352
Traded price:      4157788

Curr date is 20150210
Last traded price: 1935.86
```

```
Fluctuation ratio: -0.57%
Traded volume:     322839
Traded price:      4084514

Curr date is 20150209
Last traded price: 1947.00
Fluctuation ratio: -0.44%
Traded volume:     306825
Traded price:      4285816

Curr date is 20150206
Last traded price: 1955.52
Fluctuation ratio: +0.14%
Traded volume:     299872
Traded price:      4690502

Curr date is 20150205
Last traded price: 1952.84
Fluctuation ratio: -0.51%
Traded volume:     390988
Traded price:      4229113

Curr date is 20150204
Last traded price: 1962.79
Fluctuation ratio: +0.55%
Traded volume:     343298
Traded price:      4938163

Curr date is 20150203
Last traded price: 1951.96
Fluctuation ratio: -0.04%
Traded volume:     366309
Traded price:      4593470

Curr date is 20150202
Last traded price: 1952.68
Fluctuation ratio: +0.18%
Traded volume:     361376
Traded price:      4306479

Curr date is 20150130
Last traded price: 1949.26
Fluctuation ratio: -0.09%
Traded volume:     435570
Traded price:      5755876

Curr date is 20150129
Last traded price: 1951.02
Fluctuation ratio: -0.54%
Traded volume:     385492
Traded price:      5172438

Curr date is 20150128
```

```
Last traded price: 1961.58
Fluctuation ratio: +0.47%
Traded volume:     395675
Traded price:      5188422

Curr date is 20150127
Last traded price: 1952.40
Fluctuation ratio: +0.86%
Traded volume:     458801
Traded price:      4493085

Curr date is 20150126
Last traded price: 1935.68
Fluctuation ratio: -0.02%
Traded volume:     327120
Traded price:      3900274

Curr date is 20150123
Last traded price: 1936.09
Fluctuation ratio: +0.79%
Traded volume:     363083
Traded price:      4595883

Curr date is 20150122
Last traded price: 1920.82
Fluctuation ratio: -0.02%
Traded volume:     339567
Traded price:      4477173

Curr date is 20150121
Last traded price: 1921.23
Fluctuation ratio: +0.15%
Traded volume:     296305
Traded price:      4251709

Curr date is 20150120
Last traded price: 1918.31
Fluctuation ratio: +0.82%
Traded volume:     277640
Traded price:      3573038

Curr date is 20150119
Last traded price: 1902.62
Fluctuation ratio: +0.77%
Traded volume:     278181
Traded price:      3506180

Curr date is 20150116
Last traded price: 1888.13
Fluctuation ratio: -1.36%
Traded volume:     294638
Traded price:      3920572
```

```
Curr date is 20150115
Last traded price: 1914.14
Fluctuation ratio: +0.03%
Traded volume:     263294
Traded price:      3766993

Curr date is 20150114
Last traded price: 1913.66
Fluctuation ratio: -0.18%
Traded volume:     280976
Traded price:      4341094

Curr date is 20150113
Last traded price: 1917.14
Fluctuation ratio: -0.20%
Traded volume:     335782
Traded price:      4791776

Curr date is 20150112
Last traded price: 1920.95
Fluctuation ratio: -0.19%
Traded volume:     308845
Traded price:      3498577

Curr date is 20150109
Last traded price: 1924.70
Fluctuation ratio: +1.05%
Traded volume:     305074
Traded price:      4246382

Curr date is 20150108
Last traded price: 1904.65
Fluctuation ratio: +1.11%
Traded volume:     258491
Traded price:      4170345

Curr date is 20150107
Last traded price: 1883.83
Fluctuation ratio: +0.07%
Traded volume:     277346
Traded price:      3697278

Curr date is 20150106
Last traded price: 1882.45
Fluctuation ratio: -1.74%
Traded volume:     299342
Traded price:      4655218

Curr date is 20150105
Last traded price: 1915.75
Fluctuation ratio: -0.55%
Traded volume:     310548
Traded price:      4663122
```

```
Curr date is 20150102
Last traded price: 1926.44
Fluctuation ratio: +0.57%
Traded volume:     255350
Traded price:      3620080

Curr date is 20141230
Last traded price: 1915.59
Fluctuation ratio: -0.64%
Traded volume:     253023
Traded price:      3643011

Curr date is 20141229
Last traded price: 1927.86
Fluctuation ratio: -1.04%
Traded volume:     263159
Traded price:      3395917

Curr date is 20141226
Last traded price: 1948.16
Fluctuation ratio: +0.08%
Traded volume:     292444
Traded price:      3356911

Curr date is 20141224
Last traded price: 1946.61
Fluctuation ratio: +0.39%
Traded volume:     279433
Traded price:      2936851

Curr date is 20141223
Last traded price: 1939.02
Fluctuation ratio: -0.21%
Traded volume:     295296
Traded price:      3595299

Curr date is 20141222
Last traded price: 1943.12
Fluctuation ratio: +0.68%
Traded volume:     316460
Traded price:      4034983

Curr date is 20141219
Last traded price: 1929.98
Fluctuation ratio: +1.71%
Traded volume:     286590
Traded price:      4309175

Curr date is 20141218
Last traded price: 1897.50
Fluctuation ratio: -0.14%
Traded volume:     332349
```

```
Traded price:      5242642

Curr date is 20141217
Last traded price: 1900.16
Fluctuation ratio: -0.21%
Traded volume:     301895
Traded price:      4011260

Curr date is 20141216
Last traded price: 1904.13
Fluctuation ratio: -0.85%
Traded volume:     363279
Traded price:      4218492

Curr date is 20141215
Last traded price: 1920.36
Fluctuation ratio: -0.07%
Traded volume:     289701
Traded price:      3695248

Curr date is 20141212
Last traded price: 1921.71
Fluctuation ratio: +0.27%
Traded volume:     358699
Traded price:      3738551

Curr date is 20141211
Last traded price: 1916.59
Fluctuation ratio: -1.49%
Traded volume:     335985
Traded price:      4311388

Curr date is 20141210
Last traded price: 1945.56
Fluctuation ratio: -1.29%
Traded volume:     360982
Traded price:      3748832

Curr date is 20141209
Last traded price: 1970.95
Fluctuation ratio: -0.40%
Traded volume:     517079
Traded price:      3631508

Curr date is 20141208
Last traded price: 1978.95
Fluctuation ratio: -0.39%
Traded volume:     289019
Traded price:      3056601

Curr date is 20141205
Last traded price: 1986.62
Fluctuation ratio: +0.00%
```

```
Traded volume:     280448
Traded price:      3017469

Curr date is 20141204
Last traded price: 1986.61
Fluctuation ratio: +0.85%
Traded volume:     241669
Traded price:      3464610

Curr date is 20141203
Last traded price: 1969.91
Fluctuation ratio: +0.21%
Traded volume:     264661
Traded price:      3136377

Curr date is 20141202
Last traded price: 1965.83
Fluctuation ratio: +0.03%
Traded volume:     310614
Traded price:      3708872

Curr date is 20141201
Last traded price: 1965.22
Fluctuation ratio: -0.79%
Traded volume:     328955
Traded price:      4605221

Curr date is 20141128
Last traded price: 1980.78
Fluctuation ratio: -0.07%
Traded volume:     335862
Traded price:      4220597

Curr date is 20141127
Last traded price: 1982.09
Fluctuation ratio: +0.06%
Traded volume:     278397
Traded price:      5176725

Curr date is 20141126
Last traded price: 1980.84
Fluctuation ratio: +0.03%
Traded volume:     245777
Traded price:      3949993

Curr date is 20141125
Last traded price: 1980.21
Fluctuation ratio: +0.08%
Traded volume:     279206
Traded price:      6024583

Curr date is 20141124
Last traded price: 1978.54
```

```
Fluctuation ratio: +0.70%
Traded volume:     322592
Traded price:      5477263

Curr date is 20141121
Last traded price: 1964.84
Fluctuation ratio: +0.35%
Traded volume:     341882
Traded price:      4175043

Curr date is 20141120
Last traded price: 1958.04
Fluctuation ratio: -0.45%
Traded volume:     331852
Traded price:      4194488

Curr date is 20141119
Last traded price: 1966.87
Fluctuation ratio: -0.01%
Traded volume:     274000
Traded price:      4396794

Curr date is 20141118
Last traded price: 1967.01
Fluctuation ratio: +1.20%
Traded volume:     243905
Traded price:      4086229

Curr date is 20141117
Last traded price: 1943.63
Fluctuation ratio: -0.08%
Traded volume:     274525
Traded price:      4255101

Curr date is 20141114
Last traded price: 1945.14
Fluctuation ratio: -0.78%
Traded volume:     307503
Traded price:      4950405

Curr date is 20141113
Last traded price: 1960.51
Fluctuation ratio: -0.34%
Traded volume:     327648
Traded price:      3902488

Curr date is 20141112
Last traded price: 1967.27
Fluctuation ratio: +0.22%
Traded volume:     344719
Traded price:      4401132

Curr date is 20141111
```

```
Last traded price: 1963.00
Fluctuation ratio: +0.24%
Traded volume:     308287
Traded price:      4408109

Curr date is 20141110
Last traded price: 1958.23
Fluctuation ratio: +0.95%
Traded volume:     308267
Traded price:      4422050

Curr date is 20141107
Last traded price: 1939.87
Fluctuation ratio: +0.18%
Traded volume:     250598
Traded price:      3574807

Curr date is 20141106
Last traded price: 1936.48
Fluctuation ratio: +0.26%
Traded volume:     344538
Traded price:      4368937

Curr date is 20141105
Last traded price: 1931.43
Fluctuation ratio: -0.19%
Traded volume:     327924
Traded price:      4001394

Curr date is 20141104
Last traded price: 1935.19
Fluctuation ratio: -0.91%
Traded volume:     387726
Traded price:      4868365

Curr date is 20141103
Last traded price: 1952.97
Fluctuation ratio: -0.58%
Traded volume:     340854
Traded price:      4397324

Curr date is 20141031
Last traded price: 1964.43
Fluctuation ratio: +0.28%
Traded volume:     334087
Traded price:      5245338

Curr date is 20141030
Last traded price: 1958.93
Fluctuation ratio: -0.11%
Traded volume:     307569
Traded price:      4923400
```

```
Curr date is 20141029
Last traded price: 1961.17
Fluctuation ratio: +1.84%
Traded volume:     391852
Traded price:      5341254

Curr date is 20141028
Last traded price: 1925.68
Fluctuation ratio: -0.33%
Traded volume:     404793
Traded price:      3713021

Curr date is 20141027
Last traded price: 1931.97
Fluctuation ratio: +0.33%
Traded volume:     403265
Traded price:      3995164

Curr date is 20141024
Last traded price: 1925.69
Fluctuation ratio: -0.31%
Traded volume:     391879
Traded price:      4255770

Curr date is 20141023
Last traded price: 1931.65
Fluctuation ratio: -0.27%
Traded volume:     365647
Traded price:      4279797

Curr date is 20141022
Last traded price: 1936.97
Fluctuation ratio: +1.13%
Traded volume:     391362
Traded price:      4359710

Curr date is 20141021
Last traded price: 1915.28
Fluctuation ratio: -0.77%
Traded volume:     397515
Traded price:      4274846

Curr date is 20141020
Last traded price: 1930.06
Fluctuation ratio: +1.55%
Traded volume:     339378
Traded price:      3518148

Curr date is 20141017
Last traded price: 1900.66
Fluctuation ratio: -0.95%
Traded volume:     355535
Traded price:      4454047
```

```
Curr date is 20141016
Last traded price: 1918.83
Fluctuation ratio: -0.37%
Traded volume:     302100
Traded price:      4423301

Curr date is 20141015
Last traded price: 1925.91
Fluctuation ratio: -0.17%
Traded volume:     306734
Traded price:      4022416

Curr date is 20141014
Last traded price: 1929.25
Fluctuation ratio: +0.11%
Traded volume:     332476
Traded price:      4117644

Curr date is 20141013
Last traded price: 1927.21
Fluctuation ratio: -0.71%
Traded volume:     305428
Traded price:      3945695

Curr date is 20141010
Last traded price: 1940.92
Fluctuation ratio: -1.24%
Traded volume:     338162
Traded price:      4350972

Curr date is 20141008
Last traded price: 1965.25
Fluctuation ratio: -0.39%
Traded volume:     262800
Traded price:      3690748

Curr date is 20141007
Last traded price: 1972.91
Fluctuation ratio: +0.23%
Traded volume:     324339
Traded price:      3777823

Curr date is 20141006
Last traded price: 1968.39
Fluctuation ratio: -0.39%
Traded volume:     315350
Traded price:      4196895

Curr date is 20141002
Last traded price: 1976.16
Fluctuation ratio: -0.77%
Traded volume:     319353
```

```
Traded price:      4637472

Curr date is 20141001
Last traded price: 1991.54
Fluctuation ratio: -1.41%
Traded volume:     347820
Traded price:      4293502

Curr date is 20140930
Last traded price: 2020.09
Fluctuation ratio: -0.32%
Traded volume:     315333
Traded price:      4160112

Curr date is 20140929
Last traded price: 2026.60
Fluctuation ratio: -0.25%
Traded volume:     334737
Traded price:      4377072

Curr date is 20140926
Last traded price: 2031.64
Fluctuation ratio: -0.12%
Traded volume:     305575
Traded price:      3770029

Curr date is 20140925
Last traded price: 2034.11
Fluctuation ratio: -0.08%
Traded volume:     316799
Traded price:      4275060

Curr date is 20140924
Last traded price: 2035.64
Fluctuation ratio: +0.33%
Traded volume:     328759
Traded price:      4342212

Curr date is 20140923
Last traded price: 2028.91
Fluctuation ratio: -0.51%
Traded volume:     422689
Traded price:      4572823

Curr date is 20140922
Last traded price: 2039.27
Fluctuation ratio: -0.71%
Traded volume:     319494
Traded price:      3483751

Curr date is 20140919
Last traded price: 2053.82
Fluctuation ratio: +0.30%
```

```
Traded volume:     375520
Traded price:      4812315

Curr date is 20140918
Last traded price: 2047.74
Fluctuation ratio: -0.72%
Traded volume:     390545
Traded price:      5215264

Curr date is 20140917
Last traded price: 2062.61
Fluctuation ratio: +0.96%
Traded volume:     346442
Traded price:      3429398

Curr date is 20140916
Last traded price: 2042.92
Fluctuation ratio: +0.35%
Traded volume:     362555
Traded price:      3071310

Curr date is 20140915
Last traded price: 2035.82
Fluctuation ratio: -0.30%
Traded volume:     294911
Traded price:      3122884

Curr date is 20140912
Last traded price: 2041.86
Fluctuation ratio: +0.38%
Traded volume:     358471
Traded price:      3779245

Curr date is 20140911
Last traded price: 2034.16
Fluctuation ratio: -0.74%
Traded volume:     328302
Traded price:      4910270

Curr date is 20140905
Last traded price: 2049.41
Fluctuation ratio: -0.33%
Traded volume:     338882
Traded price:      3857244

Curr date is 20140904
Last traded price: 2056.26
Fluctuation ratio: +0.25%
Traded volume:     372306
Traded price:      4448018

Curr date is 20140903
Last traded price: 2051.20
```

```
Fluctuation ratio: -0.02%
Traded volume:     333233
Traded price:      4625007

Curr date is 20140902
Last traded price: 2051.58
Fluctuation ratio: -0.79%
Traded volume:     319591
Traded price:      3981623

Curr date is 20140901
Last traded price: 2067.86
Fluctuation ratio: -0.03%
Traded volume:     269659
Traded price:      3181819

Curr date is 20140829
Last traded price: 2068.54
Fluctuation ratio: -0.35%
Traded volume:     296756
Traded price:      3990195

Curr date is 20140828
Last traded price: 2075.76
Fluctuation ratio: +0.04%
Traded volume:     301700
Traded price:      3720860

Curr date is 20140827
Last traded price: 2074.93
Fluctuation ratio: +0.33%
Traded volume:     333998
Traded price:      4316348

Curr date is 20140826
Last traded price: 2068.05
Fluctuation ratio: +0.35%
Traded volume:     284881
Traded price:      3620635

Curr date is 20140825
Last traded price: 2060.89
Fluctuation ratio: +0.20%
Traded volume:     268890
Traded price:      3442655

Curr date is 20140822
Last traded price: 2056.70
Fluctuation ratio: +0.61%
Traded volume:     274117
Traded price:      3455265

Curr date is 20140821
```

```
Last traded price: 2044.21
Fluctuation ratio: -1.38%
Traded volume:     300227
Traded price:      4476954

Curr date is 20140820
Last traded price: 2072.78
Fluctuation ratio: +0.08%
Traded volume:     311042
Traded price:      4389712

Curr date is 20140819
Last traded price: 2071.14
Fluctuation ratio: +0.88%
Traded volume:     323722
Traded price:      4253648

Curr date is 20140818
Last traded price: 2053.13
Fluctuation ratio: -0.49%
Traded volume:     295528
Traded price:      3565288

Curr date is 20140814
Last traded price: 2063.22
Fluctuation ratio: +0.04%
Traded volume:     267368
Traded price:      3786580

Curr date is 20140813
Last traded price: 2062.36
Fluctuation ratio: +1.02%
Traded volume:     275928
Traded price:      4160760

Curr date is 20140812
Last traded price: 2041.47
Fluctuation ratio: +0.10%
Traded volume:     308125
Traded price:      3694877

Curr date is 20140811
Last traded price: 2039.37
Fluctuation ratio: +0.41%
Traded volume:     281859
Traded price:      3435591

Curr date is 20140808
Last traded price: 2031.10
Fluctuation ratio: -1.14%
Traded volume:     347908
Traded price:      4407248
```

```
Curr date is 20140807
Last traded price: 2054.51
Fluctuation ratio: -0.30%
Traded volume:     285474
Traded price:      3683729

Curr date is 20140806
Last traded price: 2060.73
Fluctuation ratio: -0.27%
Traded volume:     357521
Traded price:      4416907

Curr date is 20140805
Last traded price: 2066.26
Fluctuation ratio: -0.68%
Traded volume:     322533
Traded price:      3784559

Curr date is 20140804
Last traded price: 2080.42
Fluctuation ratio: +0.35%
Traded volume:     250575
Traded price:      3862235

Curr date is 20140801
Last traded price: 2073.10
Fluctuation ratio: -0.15%
Traded volume:     268669
Traded price:      4928825

Curr date is 20140731
Last traded price: 2076.12
Fluctuation ratio: -0.31%
Traded volume:     379557
Traded price:      5997945

Curr date is 20140730
Last traded price: 2082.61
Fluctuation ratio: +1.00%
Traded volume:     338279
Traded price:      6426401

Curr date is 20140729
Last traded price: 2061.97
Fluctuation ratio: +0.64%
Traded volume:     401105
Traded price:      5872696

Curr date is 20140728
Last traded price: 2048.81
Fluctuation ratio: +0.74%
Traded volume:     343240
Traded price:      4544045
```

```
Curr date is 20140725
Last traded price: 2033.85
Fluctuation ratio: +0.36%
Traded volume:     267408
Traded price:      3826722

Curr date is 20140724
Last traded price: 2026.62
Fluctuation ratio: -0.08%
Traded volume:     320206
Traded price:      4542263

Curr date is 20140723
Last traded price: 2028.32
Fluctuation ratio: -0.03%
Traded volume:     345338
Traded price:      4049538

Curr date is 20140722
Last traded price: 2028.93
Fluctuation ratio: +0.52%
Traded volume:     261685
Traded price:      3006868

Curr date is 20140721
Last traded price: 2018.50
Fluctuation ratio: -0.05%
Traded volume:     271415
Traded price:      3193026

Curr date is 20140718
Last traded price: 2019.42
Fluctuation ratio: -0.07%
Traded volume:     273439
Traded price:      3238917

Curr date is 20140717
Last traded price: 2020.90
Fluctuation ratio: +0.37%
Traded volume:     318949
Traded price:      3854078

Curr date is 20140716
Last traded price: 2013.48
Fluctuation ratio: +0.04%
Traded volume:     311120
Traded price:      3893282

Curr date is 20140715
Last traded price: 2012.72
Fluctuation ratio: +0.94%
Traded volume:     356640
```

```
Traded price:      4040552

Curr date is 20140714
Last traded price: 1993.88
Fluctuation ratio: +0.26%
Traded volume:     267608
Traded price:      3071958

Curr date is 20140711
Last traded price: 1988.74
Fluctuation ratio: -0.70%
Traded volume:     272598
Traded price:      3467808

Curr date is 20140710
Last traded price: 2002.84
Fluctuation ratio: +0.12%
Traded volume:     323818
Traded price:      3580097

Curr date is 20140709
Last traded price: 2000.50
Fluctuation ratio: -0.31%
Traded volume:     269156
Traded price:      3677136

Curr date is 20140708
Last traded price: 2006.66
Fluctuation ratio: +0.08%
Traded volume:     256431
Traded price:      3369937

Curr date is 20140707
Last traded price: 2005.12
Fluctuation ratio: -0.23%
Traded volume:     225549
Traded price:      2974367

Curr date is 20140704
Last traded price: 2009.66
Fluctuation ratio: -0.07%
Traded volume:     238714
Traded price:      3036642

Curr date is 20140703
Last traded price: 2010.97
Fluctuation ratio: -0.21%
Traded volume:     256694
Traded price:      3415320

Curr date is 20140702
Last traded price: 2015.28
Fluctuation ratio: +0.81%
```

```
Traded volume:     311426
Traded price:      3713358

Curr date is 20140701
Last traded price: 1999.00
Fluctuation ratio: -0.16%
Traded volume:     235523
Traded price:      2936667

Curr date is 20140630
Last traded price: 2002.21
Fluctuation ratio: +0.69%
Traded volume:     198147
Traded price:      2870175

Curr date is 20140627
Last traded price: 1988.51
Fluctuation ratio: -0.33%
Traded volume:     213631
Traded price:      2962209

Curr date is 20140626
Last traded price: 1995.05
Fluctuation ratio: +0.67%
Traded volume:     266464
Traded price:      3793358

Curr date is 20140625
Last traded price: 1981.77
Fluctuation ratio: -0.63%
Traded volume:     329627
Traded price:      3306330

Curr date is 20140624
Last traded price: 1994.35
Fluctuation ratio: +0.98%
Traded volume:     241511
Traded price:      3552323

Curr date is 20140623
Last traded price: 1974.92
Fluctuation ratio: +0.35%
Traded volume:     221379
Traded price:      3284703

Curr date is 20140620
Last traded price: 1968.07
Fluctuation ratio: -1.20%
Traded volume:     219913
Traded price:      3977340

Curr date is 20140619
Last traded price: 1992.03
```

```
Fluctuation ratio: +0.13%
Traded volume:     251781
Traded price:      3873682

Curr date is 20140618
Last traded price: 1989.49
Fluctuation ratio: -0.60%
Traded volume:     210022
Traded price:      3187212

Curr date is 20140617
Last traded price: 2001.55
Fluctuation ratio: +0.40%
Traded volume:     232786
Traded price:      3448945

Curr date is 20140616
Last traded price: 1993.59
Fluctuation ratio: +0.14%
Traded volume:     213930
Traded price:      3309603

Curr date is 20140613
Last traded price: 1990.85
Fluctuation ratio: -1.03%
Traded volume:     220593
Traded price:      3505363

Curr date is 20140612
Last traded price: 2011.65
Fluctuation ratio: -0.15%
Traded volume:     246887
Traded price:      4146952

Curr date is 20140611
Last traded price: 2014.67
Fluctuation ratio: +0.14%
Traded volume:     234720
Traded price:      3810084

Curr date is 20140610
Last traded price: 2011.80
Fluctuation ratio: +1.09%
Traded volume:     232247
Traded price:      3837230

Curr date is 20140609
Last traded price: 1990.04
Fluctuation ratio: -0.27%
Traded volume:     207205
Traded price:      4514168

Curr date is 20140605
```

```
Last traded price: 1995.48
Fluctuation ratio: -0.65%
Traded volume:     211467
Traded price:      4404876

Curr date is 20140603
Last traded price: 2008.56
Fluctuation ratio: +0.33%
Traded volume:     261739
Traded price:      4757931

Curr date is 20140602
Last traded price: 2002.00
Fluctuation ratio: +0.35%
Traded volume:     262699
Traded price:      3104545

Curr date is 20140530
Last traded price: 1994.96
Fluctuation ratio: -0.86%
Traded volume:     242647
Traded price:      4998367

Curr date is 20140529
Last traded price: 2012.26
Fluctuation ratio: -0.24%
Traded volume:     226077
Traded price:      3628824

Curr date is 20140528
Last traded price: 2017.06
Fluctuation ratio: +0.97%
Traded volume:     207463
Traded price:      3218728

Curr date is 20140527
Last traded price: 1997.63
Fluctuation ratio: -0.63%
Traded volume:     219073
Traded price:      2704692

Curr date is 20140526
Last traded price: 2010.35
Fluctuation ratio: -0.34%
Traded volume:     194315
Traded price:      2651957

Curr date is 20140523
Last traded price: 2017.17
Fluctuation ratio: +0.08%
Traded volume:     239157
Traded price:      3133989
```

```
Curr date is 20140522
Last traded price: 2015.59
Fluctuation ratio: +0.36%
Traded volume:     227074
Traded price:      3574341

Curr date is 20140521
Last traded price: 2008.33
Fluctuation ratio: -0.15%
Traded volume:     218888
Traded price:      3217907

Curr date is 20140520
Last traded price: 2011.26
Fluctuation ratio: -0.19%
Traded volume:     242307
Traded price:      3661335

Curr date is 20140519
Last traded price: 2015.14
Fluctuation ratio: +0.08%
Traded volume:     258600
Traded price:      4136234

Curr date is 20140516
Last traded price: 2013.44
Fluctuation ratio: +0.16%
Traded volume:     229481
Traded price:      3966285

Curr date is 20140515
Last traded price: 2010.20
Fluctuation ratio: -0.03%
Traded volume:     253428
Traded price:      3301753

Curr date is 20140514
Last traded price: 2010.83
Fluctuation ratio: +1.41%
Traded volume:     229158
Traded price:      3636543

Curr date is 20140513
Last traded price: 1982.93
Fluctuation ratio: +0.92%
Traded volume:     238319
Traded price:      3698504

Curr date is 20140512
Last traded price: 1964.94
Fluctuation ratio: +0.43%
Traded volume:     197864
Traded price:      3718159
```

```
Curr date is 20140509
Last traded price: 1956.55
Fluctuation ratio: +0.31%
Traded volume:     201103
Traded price:      3414683

Curr date is 20140508
Last traded price: 1950.60
Fluctuation ratio: +0.55%
Traded volume:     218940
Traded price:      3818329

Curr date is 20140507
Last traded price: 1939.88
Fluctuation ratio: -1.00%
Traded volume:     189638
Traded price:      3539128

Curr date is 20140502
Last traded price: 1959.44
Fluctuation ratio: -0.12%
Traded volume:     170004
Traded price:      3205069

Curr date is 20140430
Last traded price: 1961.79
Fluctuation ratio: -0.15%
Traded volume:     224144
Traded price:      3844149

Curr date is 20140429
Last traded price: 1964.77
Fluctuation ratio: -0.23%
Traded volume:     171640
Traded price:      2935333

Curr date is 20140428
Last traded price: 1969.26
Fluctuation ratio: -0.12%
Traded volume:     165150
Traded price:      2791504

Curr date is 20140425
Last traded price: 1971.66
Fluctuation ratio: -1.34%
Traded volume:     201771
Traded price:      3600000

Curr date is 20140424
Last traded price: 1998.34
Fluctuation ratio: -0.10%
Traded volume:     227923
```

```
Traded price:      3513555

Curr date is 20140423
Last traded price: 2000.37
Fluctuation ratio: -0.19%
Traded volume:     231735
Traded price:      3822075

Curr date is 20140422
Last traded price: 2004.22
Fluctuation ratio: +0.25%
Traded volume:     202451
Traded price:      3020345

Curr date is 20140421
Last traded price: 1999.22
Fluctuation ratio: -0.25%
Traded volume:     201221
Traded price:      2197825

Curr date is 20140418
Last traded price: 2004.28
Fluctuation ratio: +0.61%
Traded volume:     201862
Traded price:      2722967

Curr date is 20140417
Last traded price: 1992.05
Fluctuation ratio: -0.01%
Traded volume:     195693
Traded price:      2857523

Curr date is 20140416
Last traded price: 1992.21
Fluctuation ratio: -0.00%
Traded volume:     224325
Traded price:      3227487

Curr date is 20140415
Last traded price: 1992.27
Fluctuation ratio: -0.24%
Traded volume:     191446
Traded price:      3409027

Curr date is 20140414
Last traded price: 1997.02
Fluctuation ratio: -0.02%
Traded volume:     201425
Traded price:      2821669

Curr date is 20140411
Last traded price: 1997.44
Fluctuation ratio: -0.56%
```

```
Traded volume:     204566
Traded price:      3428399

Curr date is 20140410
Last traded price: 2008.61
Fluctuation ratio: +0.48%
Traded volume:     218950
Traded price:      4101410

Curr date is 20140409
Last traded price: 1998.95
Fluctuation ratio: +0.30%
Traded volume:     230827
Traded price:      4695200

Curr date is 20140408
Last traded price: 1993.03
Fluctuation ratio: +0.17%
Traded volume:     214842
Traded price:      3702218

Curr date is 20140407
Last traded price: 1989.70
Fluctuation ratio: +0.08%
Traded volume:     182372
Traded price:      3590499

Curr date is 20140404
Last traded price: 1988.09
Fluctuation ratio: -0.28%
Traded volume:     212662
Traded price:      3215389

Curr date is 20140403
Last traded price: 1993.70
Fluctuation ratio: -0.18%
Traded volume:     222076
Traded price:      4122174

Curr date is 20140402
Last traded price: 1997.25
Fluctuation ratio: +0.26%
Traded volume:     217813
Traded price:      3905603

Curr date is 20140401
Last traded price: 1991.98
Fluctuation ratio: +0.32%
Traded volume:     208181
Traded price:      3708410

Curr date is 20140331
Last traded price: 1985.61
```

```
Fluctuation ratio: +0.23%
Traded volume:     237572
Traded price:      4391688

Curr date is 20140328
Last traded price: 1981.00
Fluctuation ratio: +0.15%
Traded volume:     241713
Traded price:      3392042

Curr date is 20140327
Last traded price: 1977.97
Fluctuation ratio: +0.70%
Traded volume:     212688
Traded price:      3867323

Curr date is 20140326
Last traded price: 1964.31
Fluctuation ratio: +1.19%
Traded volume:     243349
Traded price:      3972279

Curr date is 20140325
Last traded price: 1941.25
Fluctuation ratio: -0.22%
Traded volume:     222621
Traded price:      3728658

Curr date is 20140324
Last traded price: 1945.55
Fluctuation ratio: +0.55%
Traded volume:     225732
Traded price:      3264262

Curr date is 20140321
Last traded price: 1934.94
Fluctuation ratio: +0.80%
Traded volume:     245724
Traded price:      3373507

Curr date is 20140320
Last traded price: 1919.52
Fluctuation ratio: -0.94%
Traded volume:     216319
Traded price:      3275680

Curr date is 20140319
Last traded price: 1937.68
Fluctuation ratio: -0.13%
Traded volume:     246298
Traded price:      3363229

Curr date is 20140318
```

```
Last traded price: 1940.21
Fluctuation ratio: +0.66%
Traded volume:     270588
Traded price:      3560038

Curr date is 20140317
Last traded price: 1927.53
Fluctuation ratio: +0.40%
Traded volume:     212146
Traded price:      3264799

Curr date is 20140314
Last traded price: 1919.90
Fluctuation ratio: -0.75%
Traded volume:     251334
Traded price:      3976822

Curr date is 20140313
Last traded price: 1934.38
Fluctuation ratio: +0.10%
Traded volume:     256319
Traded price:      4349817

Curr date is 20140312
Last traded price: 1932.54
Fluctuation ratio: -1.60%
Traded volume:     316494
Traded price:      4125029

Curr date is 20140311
Last traded price: 1963.87
Fluctuation ratio: +0.48%
Traded volume:     244704
Traded price:      3387413

Curr date is 20140310
Last traded price: 1954.42
Fluctuation ratio: -1.03%
Traded volume:     207689
Traded price:      3351209

Curr date is 20140307
Last traded price: 1974.68
Fluctuation ratio: -0.05%
Traded volume:     224665
Traded price:      3208114

Curr date is 20140306
Last traded price: 1975.62
Fluctuation ratio: +0.22%
Traded volume:     188961
Traded price:      3301684
```

```
Curr date is 20140305
Last traded price: 1971.24
Fluctuation ratio: +0.88%
Traded volume:     194812
Traded price:      3567045

Curr date is 20140304
Last traded price: 1954.11
Fluctuation ratio: -0.54%
Traded volume:     197609
Traded price:      3036204

Curr date is 20140303
Last traded price: 1964.69
Fluctuation ratio: -0.77%
Traded volume:     220309
Traded price:      3334422

Curr date is 20140228
Last traded price: 1979.99
Fluctuation ratio: +0.08%
Traded volume:     245561
Traded price:      4724885

Curr date is 20140227
Last traded price: 1978.43
Fluctuation ratio: +0.39%
Traded volume:     198142
Traded price:      3763740

Curr date is 20140226
Last traded price: 1970.77
Fluctuation ratio: +0.30%
Traded volume:     204839
Traded price:      3785678

Curr date is 20140225
Last traded price: 1964.86
Fluctuation ratio: +0.81%
Traded volume:     222865
Traded price:      3864116

Curr date is 20140224
Last traded price: 1949.05
Fluctuation ratio: -0.45%
Traded volume:     191774
Traded price:      2830628

Curr date is 20140221
Last traded price: 1957.83
Fluctuation ratio: +1.41%
Traded volume:     209495
Traded price:      3455116
```

```
Curr date is 20140220
Last traded price: 1930.57
Fluctuation ratio: -0.64%
Traded volume:     228428
Traded price:      3787085

Curr date is 20140219
Last traded price: 1942.93
Fluctuation ratio: -0.20%
Traded volume:     208354
Traded price:      2991546

Curr date is 20140218
Last traded price: 1946.91
Fluctuation ratio: +0.03%
Traded volume:     198390
Traded price:      2689169

Curr date is 20140217
Last traded price: 1946.36
Fluctuation ratio: +0.31%
Traded volume:     211950
Traded price:      3353405

Curr date is 20140214
Last traded price: 1940.28
Fluctuation ratio: +0.69%
Traded volume:     212453
Traded price:      3158472

Curr date is 20140213
Last traded price: 1926.96
Fluctuation ratio: -0.46%
Traded volume:     183349
Traded price:      2865046

Curr date is 20140212
Last traded price: 1935.84
Fluctuation ratio: +0.20%
Traded volume:     222257
Traded price:      3569015

Curr date is 20140211
Last traded price: 1932.06
Fluctuation ratio: +0.46%
Traded volume:     217817
Traded price:      3187798

Curr date is 20140210
Last traded price: 1923.30
Fluctuation ratio: +0.04%
Traded volume:     205334
```

```
Traded price:      3154436

Curr date is 20140207
Last traded price: 1922.50
Fluctuation ratio: +0.77%
Traded volume:     270152
Traded price:      4343953

Curr date is 20140206
Last traded price: 1907.89
Fluctuation ratio: +0.88%
Traded volume:     255198
Traded price:      3940198

Curr date is 20140205
Last traded price: 1891.32
Fluctuation ratio: +0.24%
Traded volume:     240719
Traded price:      3889316

Curr date is 20140204
Last traded price: 1886.85
Fluctuation ratio: -1.72%
Traded volume:     246724
Traded price:      4427814

Curr date is 20140203
Last traded price: 1919.96
Fluctuation ratio: -1.09%
Traded volume:     226810
Traded price:      3903330

Curr date is 20140129
Last traded price: 1941.15
Fluctuation ratio: +1.26%
Traded volume:     213123
Traded price:      4182197

Curr date is 20140128
Last traded price: 1916.93
Fluctuation ratio: +0.34%
Traded volume:     267292
Traded price:      4130271

Curr date is 20140127
Last traded price: 1910.34
Fluctuation ratio: -1.56%
Traded volume:     336281
Traded price:      4321557

Curr date is 20140124
Last traded price: 1940.56
Fluctuation ratio: -0.36%
```

```
Traded volume:     275118
Traded price:      4301718

Curr date is 20140123
Last traded price: 1947.59
Fluctuation ratio: -1.16%
Traded volume:     222460
Traded price:      3380935

Curr date is 20140122
Last traded price: 1970.42
Fluctuation ratio: +0.33%
Traded volume:     236308
Traded price:      3405786

Curr date is 20140121
Last traded price: 1963.89
Fluctuation ratio: +0.52%
Traded volume:     286305
Traded price:      3299944

Curr date is 20140120
Last traded price: 1953.78
Fluctuation ratio: +0.48%
Traded volume:     275147
Traded price:      2818977

Curr date is 20140117
Last traded price: 1944.48
Fluctuation ratio: -0.66%
Traded volume:     257056
Traded price:      3256670

Curr date is 20140116
Last traded price: 1957.32
Fluctuation ratio: +0.21%
Traded volume:     373252
Traded price:      3317219

Curr date is 20140115
Last traded price: 1953.28
Fluctuation ratio: +0.37%
Traded volume:     247951
Traded price:      3757071

Curr date is 20140114
Last traded price: 1946.07
Fluctuation ratio: -0.15%
Traded volume:     241247
Traded price:      3448476

Curr date is 20140113
Last traded price: 1948.92
```

```
Fluctuation ratio: +0.54%
Traded volume:     264162
Traded price:      3292457

Curr date is 20140110
Last traded price: 1938.54
Fluctuation ratio: -0.39%
Traded volume:     254220
Traded price:      4213852

Curr date is 20140109
Last traded price: 1946.11
Fluctuation ratio: -0.66%
Traded volume:     272051
Traded price:      4584640

Curr date is 20140108
Last traded price: 1958.96
Fluctuation ratio: -0.02%
Traded volume:     214449
Traded price:      4244571

Curr date is 20140107
Last traded price: 1959.44
Fluctuation ratio: +0.32%
Traded volume:     190804
Traded price:      3606740

Curr date is 20140106
Last traded price: 1953.28
Fluctuation ratio: +0.37%
Traded volume:     191204
Traded price:      3594316

Curr date is 20140103
Last traded price: 1946.14
Fluctuation ratio: -1.07%
Traded volume:     186078
Traded price:      3919151

Curr date is 20140102
Last traded price: 1967.19
Fluctuation ratio: -2.20%
Traded volume:     204371
Traded price:      4222610

Curr date is 20131230
Last traded price: 2011.34
Fluctuation ratio: +0.45%
Traded volume:     167783
Traded price:      3174283

Curr date is 20131227
```

```
Last traded price: 2002.28
Fluctuation ratio: +0.15%
Traded volume:     202975
Traded price:      2894294

Curr date is 20131226
Last traded price: 1999.30
Fluctuation ratio: -0.11%
Traded volume:     215248
Traded price:      3164040

Curr date is 20131224
Last traded price: 2001.59
Fluctuation ratio: +0.24%
Traded volume:     204456
Traded price:      3072365

Curr date is 20131223
Last traded price: 1996.89
Fluctuation ratio: +0.68%
Traded volume:     218569
Traded price:      3176738

Curr date is 20131220
Last traded price: 1983.35
Fluctuation ratio: +0.39%
Traded volume:     237190
Traded price:      3056046

Curr date is 20131219
Last traded price: 1975.65
Fluctuation ratio: +0.05%
Traded volume:     251453
Traded price:      3717188

Curr date is 20131218
Last traded price: 1974.63
Fluctuation ratio: +0.45%
Traded volume:     254192
Traded price:      3688395

Curr date is 20131217
Last traded price: 1965.74
Fluctuation ratio: +0.23%
Traded volume:     222303
Traded price:      3141933

Curr date is 20131216
Last traded price: 1961.15
Fluctuation ratio: -0.09%
Traded volume:     210213
Traded price:      2809840
```

```
Curr date is 20131213
Last traded price: 1962.91
Fluctuation ratio: -0.26%
Traded volume:     220057
Traded price:      3230596

Curr date is 20131212
Last traded price: 1967.93
Fluctuation ratio: -0.51%
Traded volume:     225760
Traded price:      3550659

Curr date is 20131211
Last traded price: 1977.97
Fluctuation ratio: -0.78%
Traded volume:     228492
Traded price:      3374674

Curr date is 20131210
Last traded price: 1993.45
Fluctuation ratio: -0.35%
Traded volume:     195872
Traded price:      2939856

Curr date is 20131209
Last traded price: 2000.38
Fluctuation ratio: +1.01%
Traded volume:     221857
Traded price:      2962442

Curr date is 20131206
Last traded price: 1980.41
Fluctuation ratio: -0.22%
Traded volume:     198154
Traded price:      3229785

Curr date is 20131205
Last traded price: 1984.77
Fluctuation ratio: -0.10%
Traded volume:     213097
Traded price:      3254604

Curr date is 20131204
Last traded price: 1986.80
Fluctuation ratio: -1.12%
Traded volume:     223651
Traded price:      3394554

Curr date is 20131203
Last traded price: 2009.36
Fluctuation ratio: -1.05%
Traded volume:     290721
Traded price:      3591374
```

```
Curr date is 20131202
Last traded price: 2030.78
Fluctuation ratio: -0.69%
Traded volume:     224760
Traded price:      3201990

Curr date is 20131129
Last traded price: 2044.87
Fluctuation ratio: -0.04%
Traded volume:     241045
Traded price:      3348871

Curr date is 20131128
Last traded price: 2045.77
Fluctuation ratio: +0.84%
Traded volume:     270438
Traded price:      3994147

Curr date is 20131127
Last traded price: 2028.81
Fluctuation ratio: +0.31%
Traded volume:     252787
Traded price:      3630218

Curr date is 20131126
Last traded price: 2022.64
Fluctuation ratio: +0.33%
Traded volume:     270419
Traded price:      3400474

Curr date is 20131125
Last traded price: 2015.98
Fluctuation ratio: +0.49%
Traded volume:     284901
Traded price:      2984885

Curr date is 20131122
Last traded price: 2006.23
Fluctuation ratio: +0.62%
Traded volume:     343239
Traded price:      3377985

Curr date is 20131121
Last traded price: 1993.78
Fluctuation ratio: -1.16%
Traded volume:     281466
Traded price:      3241605

Curr date is 20131120
Last traded price: 2017.24
Fluctuation ratio: -0.71%
Traded volume:     310754
```

```
Traded price:      3737041

Curr date is 20131119
Last traded price: 2031.64
Fluctuation ratio: +1.04%
Traded volume:     397334
Traded price:      4232086

Curr date is 20131118
Last traded price: 2010.81
Fluctuation ratio: +0.26%
Traded volume:     310238
Traded price:      3023580

Curr date is 20131115
Last traded price: 2005.64
Fluctuation ratio: +1.94%
Traded volume:     259971
Traded price:      3612950

Curr date is 20131114
Last traded price: 1967.56
Fluctuation ratio: +0.20%
Traded volume:     319306
Traded price:      3691958

Curr date is 20131113
Last traded price: 1963.56
Fluctuation ratio: -1.60%
Traded volume:     246137
Traded price:      3340619

Curr date is 20131112
Last traded price: 1995.48
Fluctuation ratio: +0.92%
Traded volume:     242746
Traded price:      3154383

Curr date is 20131111
Last traded price: 1977.30
Fluctuation ratio: -0.38%
Traded volume:     282669
Traded price:      3258500

Curr date is 20131108
Last traded price: 1984.87
Fluctuation ratio: -0.96%
Traded volume:     272045
Traded price:      3843067

Curr date is 20131107
Last traded price: 2004.04
Fluctuation ratio: -0.48%
```

```
Traded volume:     307285
Traded price:      3617354

Curr date is 20131106
Last traded price: 2013.67
Fluctuation ratio: -0.01%
Traded volume:     267841
Traded price:      3881236

Curr date is 20131105
Last traded price: 2013.93
Fluctuation ratio: -0.56%
Traded volume:     254851
Traded price:      3480379

Curr date is 20131104
Last traded price: 2025.17
Fluctuation ratio: -0.70%
Traded volume:     232966
Traded price:      3479051

Curr date is 20131101
Last traded price: 2039.42
Fluctuation ratio: +0.46%
Traded volume:     290539
Traded price:      3567554

Curr date is 20131031
Last traded price: 2030.09
Fluctuation ratio: -1.43%
Traded volume:     251008
Traded price:      4403615

Curr date is 20131030
Last traded price: 2059.58
Fluctuation ratio: +0.38%
Traded volume:     258990
Traded price:      4055160

Curr date is 20131029
Last traded price: 2051.76
Fluctuation ratio: +0.18%
Traded volume:     303036
Traded price:      3536341

Curr date is 20131028
Last traded price: 2048.14
Fluctuation ratio: +0.68%
Traded volume:     252973
Traded price:      3842925

Curr date is 20131025
Last traded price: 2034.39
```

```
Fluctuation ratio: -0.60%
Traded volume:     230750
Traded price:      3956814

Curr date is 20131024
Last traded price: 2046.69
Fluctuation ratio: +0.54%
Traded volume:     251631
Traded price:      3751803

Curr date is 20131023
Last traded price: 2035.75
Fluctuation ratio: -0.99%
Traded volume:     354778
Traded price:      4862434

Curr date is 20131022
Last traded price: 2056.12
Fluctuation ratio: +0.15%
Traded volume:     370326
Traded price:      4494164

Curr date is 20131021
Last traded price: 2053.01
Fluctuation ratio: +0.03%
Traded volume:     270710
Traded price:      4073675

Curr date is 20131018
Last traded price: 2052.40
Fluctuation ratio: +0.58%
Traded volume:     252614
Traded price:      4374852

Curr date is 20131017
Last traded price: 2040.61
Fluctuation ratio: +0.29%
Traded volume:     303614
Traded price:      4835640

Curr date is 20131016
Last traded price: 2034.61
Fluctuation ratio: -0.31%
Traded volume:     263641
Traded price:      4314649

Curr date is 20131015
Last traded price: 2040.96
Fluctuation ratio: +1.02%
Traded volume:     223537
Traded price:      3985446

Curr date is 20131014
```

```
Last traded price: 2020.27
Fluctuation ratio: -0.23%
Traded volume:     218427
Traded price:      3580686

Curr date is 20131011
Last traded price: 2024.90
Fluctuation ratio: +1.17%
Traded volume:     251117
Traded price:      4493090

Curr date is 20131010
Last traded price: 2001.40
Fluctuation ratio: -0.07%
Traded volume:     219492
Traded price:      3845368

Curr date is 20131008
Last traded price: 2002.76
Fluctuation ratio: +0.42%
Traded volume:     311603
Traded price:      3672424

Curr date is 20131007
Last traded price: 1994.42
Fluctuation ratio: -0.13%
Traded volume:     246293
Traded price:      3393471

Curr date is 20131004
Last traded price: 1996.98
Fluctuation ratio: -0.12%
Traded volume:     237774
Traded price:      4253764

Curr date is 20131002
Last traded price: 1999.47
Fluctuation ratio: +0.03%
Traded volume:     234244
Traded price:      4304362

Curr date is 20131001
Last traded price: 1998.87
Fluctuation ratio: +0.10%
Traded volume:     277789
Traded price:      3833118

Curr date is 20130930
Last traded price: 1996.96
Fluctuation ratio: -0.74%
Traded volume:     285755
Traded price:      4094276
```

```
Curr date is 20130927
Last traded price: 2011.80
Fluctuation ratio: +0.22%
Traded volume:     277557
Traded price:      3999765

Curr date is 20130926
Last traded price: 2007.32
Fluctuation ratio: +0.46%
Traded volume:     250860
Traded price:      3891191

Curr date is 20130925
Last traded price: 1998.06
Fluctuation ratio: -0.45%
Traded volume:     276371
Traded price:      4078093

Curr date is 20130924
Last traded price: 2007.10
Fluctuation ratio: -0.11%
Traded volume:     305884
Traded price:      4180854

Curr date is 20130923
Last traded price: 2009.41
Fluctuation ratio: +0.19%
Traded volume:     244437
Traded price:      4899864

Curr date is 20130917
Last traded price: 2005.58
Fluctuation ratio: -0.39%
Traded volume:     259527
Traded price:      4255607

Curr date is 20130916
Last traded price: 2013.37
Fluctuation ratio: +0.96%
Traded volume:     245398
Traded price:      4059219

Curr date is 20130913
Last traded price: 1994.32
Fluctuation ratio: -0.49%
Traded volume:     246637
Traded price:      3921748

Curr date is 20130912
Last traded price: 2004.06
Fluctuation ratio: +0.01%
Traded volume:     351596
Traded price:      5673169
```

```
Curr date is 20130911
Last traded price: 2003.85
Fluctuation ratio: +0.49%
Traded volume:     295593
Traded price:      4564929

Curr date is 20130910
Last traded price: 1994.06
Fluctuation ratio: +0.98%
Traded volume:     259211
Traded price:      5270527

Curr date is 20130909
Last traded price: 1974.67
Fluctuation ratio: +0.99%
Traded volume:     265792
Traded price:      4428000

Curr date is 20130906
Last traded price: 1955.31
Fluctuation ratio: +0.19%
Traded volume:     310247
Traded price:      4162215

Curr date is 20130905
Last traded price: 1951.65
Fluctuation ratio: +0.96%
Traded volume:     282456
Traded price:      4404963

Curr date is 20130904
Last traded price: 1933.03
Fluctuation ratio: -0.04%
Traded volume:     277840
Traded price:      3728333

Curr date is 20130903
Last traded price: 1933.74
Fluctuation ratio: +0.46%
Traded volume:     293834
Traded price:      3967549

Curr date is 20130902
Last traded price: 1924.81
Fluctuation ratio: -0.08%
Traded volume:     240051
Traded price:      3372585

Curr date is 20130830
Last traded price: 1926.36
Fluctuation ratio: +0.99%
Traded volume:     296211
```

```
Traded price:      4961420

Curr date is 20130829
Last traded price: 1907.54
Fluctuation ratio: +1.22%
Traded volume:     296958
Traded price:      4705602

Curr date is 20130828
Last traded price: 1884.52
Fluctuation ratio: -0.07%
Traded volume:     372184
Traded price:      3356025

Curr date is 20130827
Last traded price: 1885.84
Fluctuation ratio: -0.11%
Traded volume:     403468
Traded price:      3719495

Curr date is 20130826
Last traded price: 1887.86
Fluctuation ratio: +0.95%
Traded volume:     312771
Traded price:      3225582

Curr date is 20130823
Last traded price: 1870.16
Fluctuation ratio: +1.14%
Traded volume:     368614
Traded price:      3804927

Curr date is 20130822
Last traded price: 1849.12
Fluctuation ratio: -0.98%
Traded volume:     421111
Traded price:      4218476

Curr date is 20130821
Last traded price: 1867.46
Fluctuation ratio: -1.08%
Traded volume:     407101
Traded price:      4250000

Curr date is 20130820
Last traded price: 1887.85
Fluctuation ratio: -1.55%
Traded volume:     346148
Traded price:      4201311

Curr date is 20130819
Last traded price: 1917.64
Fluctuation ratio: -0.13%
```

```
Traded volume:     296265
Traded price:      3230968

Curr date is 20130816
Last traded price: 1920.11
Fluctuation ratio: -0.20%
Traded volume:     343217
Traded price:      3720318

Curr date is 20130814
Last traded price: 1923.91
Fluctuation ratio: +0.57%
Traded volume:     358594
Traded price:      4065007

Curr date is 20130813
Last traded price: 1913.03
Fluctuation ratio: +1.50%
Traded volume:     358365
Traded price:      3993280

Curr date is 20130812
Last traded price: 1884.83
Fluctuation ratio: +0.22%
Traded volume:     294126
Traded price:      3030875

Curr date is 20130809
Last traded price: 1880.71
Fluctuation ratio: -0.17%
Traded volume:     276670
Traded price:      3467209

Curr date is 20130808
Last traded price: 1883.97
Fluctuation ratio: +0.30%
Traded volume:     333986
Traded price:      3599826

Curr date is 20130807
Last traded price: 1878.33
Fluctuation ratio: -1.48%
Traded volume:     308230
Traded price:      3495811

Curr date is 20130806
Last traded price: 1906.62
Fluctuation ratio: -0.50%
Traded volume:     291374
Traded price:      3451440

Curr date is 20130805
Last traded price: 1916.22
```

```
Fluctuation ratio: -0.37%
Traded volume:     231098
Traded price:      2715271

Curr date is 20130802
Last traded price: 1923.38
Fluctuation ratio: +0.14%
Traded volume:     294265
Traded price:      3970692

Curr date is 20130801
Last traded price: 1920.74
Fluctuation ratio: +0.35%
Traded volume:     330396
Traded price:      3547228

Curr date is 20130731
Last traded price: 1914.03
Fluctuation ratio: -0.16%
Traded volume:     379555
Traded price:      3998891

Curr date is 20130730
Last traded price: 1917.05
Fluctuation ratio: +0.90%
Traded volume:     362600
Traded price:      3511439

Curr date is 20130729
Last traded price: 1899.89
Fluctuation ratio: -0.57%
Traded volume:     333010
Traded price:      4165391

Curr date is 20130726
Last traded price: 1910.81
Fluctuation ratio: +0.06%
Traded volume:     316379
Traded price:      3742114

Curr date is 20130725
Last traded price: 1909.61
Fluctuation ratio: -0.13%
Traded volume:     304352
Traded price:      3834601

Curr date is 20130724
Last traded price: 1912.08
Fluctuation ratio: +0.42%
Traded volume:     316057
Traded price:      3452878

Curr date is 20130723
```

```
Last traded price: 1904.15
Fluctuation ratio: +1.27%
Traded volume:     347577
Traded price:      4246026

Curr date is 20130722
Last traded price: 1880.35
Fluctuation ratio: +0.48%
Traded volume:     353280
Traded price:      3588523

Curr date is 20130719
Last traded price: 1871.41
Fluctuation ratio: -0.22%
Traded volume:     315469
Traded price:      3549418

Curr date is 20130718
Last traded price: 1875.48
Fluctuation ratio: -0.64%
Traded volume:     376180
Traded price:      3468481

Curr date is 20130717
Last traded price: 1887.49
Fluctuation ratio: +1.13%
Traded volume:     378452
Traded price:      3969984

Curr date is 20130716
Last traded price: 1866.36
Fluctuation ratio: -0.47%
Traded volume:     300367
Traded price:      2947965

Curr date is 20130715
Last traded price: 1875.16
Fluctuation ratio: +0.28%
Traded volume:     326640
Traded price:      3154544

Curr date is 20130712
Last traded price: 1869.98
Fluctuation ratio: -0.41%
Traded volume:     262391
Traded price:      3411133

Curr date is 20130711
Last traded price: 1877.60
Fluctuation ratio: +2.93%
Traded volume:     327399
Traded price:      3914282
```

```
Curr date is 20130710
Last traded price: 1824.16
Fluctuation ratio: -0.34%
Traded volume:     264404
Traded price:      3160738

Curr date is 20130709
Last traded price: 1830.35
Fluctuation ratio: +0.74%
Traded volume:     254672
Traded price:      3114214

Curr date is 20130708
Last traded price: 1816.85
Fluctuation ratio: -0.90%
Traded volume:     299820
Traded price:      3483488

Curr date is 20130705
Last traded price: 1833.31
Fluctuation ratio: -0.32%
Traded volume:     280758
Traded price:      3721229

Curr date is 20130704
Last traded price: 1839.14
Fluctuation ratio: +0.79%
Traded volume:     309056
Traded price:      3093375

Curr date is 20130703
Last traded price: 1824.66
Fluctuation ratio: -1.64%
Traded volume:     399890
Traded price:      3654308

Curr date is 20130702
Last traded price: 1855.02
Fluctuation ratio: -0.04%
Traded volume:     479609
Traded price:      3843096

Curr date is 20130701
Last traded price: 1855.73
Fluctuation ratio: -0.41%
Traded volume:     217221
Traded price:      2937167

Curr date is 20130628
Last traded price: 1863.32
Fluctuation ratio: +1.56%
Traded volume:     246022
Traded price:      4343805
```

```
Curr date is 20130627
Last traded price: 1834.70
Fluctuation ratio: +2.87%
Traded volume:     279711
Traded price:      4052271

Curr date is 20130626
Last traded price: 1783.45
Fluctuation ratio: +0.16%
Traded volume:     293431
Traded price:      4046496

Curr date is 20130625
Last traded price: 1780.63
Fluctuation ratio: -1.02%
Traded volume:     328098
Traded price:      4410325

Curr date is 20130624
Last traded price: 1799.01
Fluctuation ratio: -1.31%
Traded volume:     217788
Traded price:      3116591

Curr date is 20130621
Last traded price: 1822.83
Fluctuation ratio: -1.49%
Traded volume:     281888
Traded price:      4891392

Curr date is 20130620
Last traded price: 1850.49
Fluctuation ratio: -2.00%
Traded volume:     350614
Traded price:      4311914

Curr date is 20130619
Last traded price: 1888.31
Fluctuation ratio: -0.65%
Traded volume:     276990
Traded price:      3278293

Curr date is 20130618
Last traded price: 1900.62
Fluctuation ratio: +0.93%
Traded volume:     275366
Traded price:      3388892

Curr date is 20130617
Last traded price: 1883.10
Fluctuation ratio: -0.32%
Traded volume:     258662
```

```
Traded price:      3192420

Curr date is 20130614
Last traded price: 1889.24
Fluctuation ratio: +0.35%
Traded volume:     275807
Traded price:      3449511

Curr date is 20130613
Last traded price: 1882.73
Fluctuation ratio: -1.42%
Traded volume:     301949
Traded price:      4957423

Curr date is 20130612
Last traded price: 1909.91
Fluctuation ratio: -0.56%
Traded volume:     250486
Traded price:      3375032

Curr date is 20130611
Last traded price: 1920.68
Fluctuation ratio: -0.62%
Traded volume:     285290
Traded price:      4379626

Curr date is 20130610
Last traded price: 1932.70
Fluctuation ratio: +0.46%
Traded volume:     268136
Traded price:      3679710

Curr date is 20130607
Last traded price: 1923.85
Fluctuation ratio: -1.80%
Traded volume:     337193
Traded price:      5434851

Curr date is 20130605
Last traded price: 1959.19
Fluctuation ratio: -1.52%
Traded volume:     335946
Traded price:      4221363

Curr date is 20130604
Last traded price: 1989.51
Fluctuation ratio: -0.00%
Traded volume:     329098
Traded price:      3889375

Curr date is 20130603
Last traded price: 1989.57
Fluctuation ratio: -0.57%
```

```
Traded volume:     319509
Traded price:      3588869

Curr date is 20130531
Last traded price: 2001.05
Fluctuation ratio: +0.05%
Traded volume:     390918
Traded price:      5274203

Curr date is 20130530
Last traded price: 2000.10
Fluctuation ratio: -0.05%
Traded volume:     423015
Traded price:      4783717

Curr date is 20130529
Last traded price: 2001.20
Fluctuation ratio: +0.75%
Traded volume:     356232
Traded price:      4657062

Curr date is 20130528
Last traded price: 1986.22
Fluctuation ratio: +0.32%
Traded volume:     298109
Traded price:      3172256

Curr date is 20130527
Last traded price: 1979.97
Fluctuation ratio: +0.33%
Traded volume:     251666
Traded price:      2927484

Curr date is 20130524
Last traded price: 1973.45
Fluctuation ratio: +0.22%
Traded volume:     268745
Traded price:      3668406

Curr date is 20130523
Last traded price: 1969.19
Fluctuation ratio: -1.24%
Traded volume:     338156
Traded price:      4527767

Curr date is 20130522
Last traded price: 1993.83
Fluctuation ratio: +0.64%
Traded volume:     322252
Traded price:      4447500

Curr date is 20130521
Last traded price: 1981.09
```

```
Fluctuation ratio: -0.07%
Traded volume:     241815
Traded price:      3630840

Curr date is 20130520
Last traded price: 1982.43
Fluctuation ratio: -0.22%
Traded volume:     270052
Traded price:      4323120

Curr date is 20130516
Last traded price: 1986.81
Fluctuation ratio: +0.79%
Traded volume:     316121
Traded price:      4590841

Curr date is 20130515
Last traded price: 1971.26
Fluctuation ratio: +0.12%
Traded volume:     240078
Traded price:      3633929

Curr date is 20130514
Last traded price: 1968.83
Fluctuation ratio: +1.03%
Traded volume:     325003
Traded price:      4065479

Curr date is 20130513
Last traded price: 1948.70
Fluctuation ratio: +0.20%
Traded volume:     245468
Traded price:      2955649

Curr date is 20130510
Last traded price: 1944.75
Fluctuation ratio: -1.75%
Traded volume:     279348
Traded price:      4116846

Curr date is 20130509
Last traded price: 1979.45
Fluctuation ratio: +1.18%
Traded volume:     326441
Traded price:      4965549

Curr date is 20130508
Last traded price: 1956.45
Fluctuation ratio: +0.11%
Traded volume:     332720
Traded price:      3964107

Curr date is 20130507
```

```
Last traded price: 1954.35
Fluctuation ratio: -0.36%
Traded volume:     296110
Traded price:      3499695

Curr date is 20130506
Last traded price: 1961.48
Fluctuation ratio: -0.22%
Traded volume:     292026
Traded price:      3894599

Curr date is 20130503
Last traded price: 1965.71
Fluctuation ratio: +0.43%
Traded volume:     307027
Traded price:      4124998

Curr date is 20130502
Last traded price: 1957.21
Fluctuation ratio: -0.34%
Traded volume:     342977
Traded price:      4037749

Curr date is 20130430
Last traded price: 1963.95
Fluctuation ratio: +1.20%
Traded volume:     398239
Traded price:      4848700

Curr date is 20130429
Last traded price: 1940.70
Fluctuation ratio: -0.20%
Traded volume:     261788
Traded price:      3597071

Curr date is 20130426
Last traded price: 1944.56
Fluctuation ratio: -0.36%
Traded volume:     274235
Traded price:      4003127

Curr date is 20130425
Last traded price: 1951.60
Fluctuation ratio: +0.84%
Traded volume:     326483
Traded price:      4407082

Curr date is 20130424
Last traded price: 1935.31
Fluctuation ratio: +0.87%
Traded volume:     335779
Traded price:      4363872
```

```
Curr date is 20130423
Last traded price: 1918.63
Fluctuation ratio: -0.40%
Traded volume:     254517
Traded price:      3458067

Curr date is 20130422
Last traded price: 1926.31
Fluctuation ratio: +1.03%
Traded volume:     255781
Traded price:      3626720

Curr date is 20130419
Last traded price: 1906.75
Fluctuation ratio: +0.35%
Traded volume:     326680
Traded price:      4379291

Curr date is 20130418
Last traded price: 1900.06
Fluctuation ratio: -1.24%
Traded volume:     338092
Traded price:      4206375

Curr date is 20130417
Last traded price: 1923.84
Fluctuation ratio: +0.08%
Traded volume:     409459
Traded price:      5264676

Curr date is 20130416
Last traded price: 1922.21
Fluctuation ratio: +0.09%
Traded volume:     380317
Traded price:      4777542

Curr date is 20130415
Last traded price: 1920.45
Fluctuation ratio: -0.20%
Traded volume:     327644
Traded price:      4215041

Curr date is 20130412
Last traded price: 1924.23
Fluctuation ratio: -1.31%
Traded volume:     401063
Traded price:      4902446

Curr date is 20130411
Last traded price: 1949.80
Fluctuation ratio: +0.73%
Traded volume:     381793
Traded price:      4720942
```

```
Curr date is 20130410
Last traded price: 1935.58
Fluctuation ratio: +0.77%
Traded volume:     314291
Traded price:      4090616

Curr date is 20130409
Last traded price: 1920.74
Fluctuation ratio: +0.11%
Traded volume:     350146
Traded price:      3816736

Curr date is 20130408
Last traded price: 1918.69
Fluctuation ratio: -0.44%
Traded volume:     306636
Traded price:      3882161

Curr date is 20130405
Last traded price: 1927.23
Fluctuation ratio: -1.64%
Traded volume:     366655
Traded price:      4850932

Curr date is 20130404
Last traded price: 1959.45
Fluctuation ratio: -1.20%
Traded volume:     410066
Traded price:      4640638

Curr date is 20130403
Last traded price: 1983.22
Fluctuation ratio: -0.15%
Traded volume:     387339
Traded price:      4069098

Curr date is 20130402
Last traded price: 1986.15
Fluctuation ratio: -0.49%
Traded volume:     490325
Traded price:      4392183

Curr date is 20130401
Last traded price: 1995.99
Fluctuation ratio: -0.44%
Traded volume:     264572
Traded price:      2577942

Curr date is 20130329
Last traded price: 2004.89
Fluctuation ratio: +0.57%
Traded volume:     338522
```

```
Traded price:      3444653

Curr date is 20130328
Last traded price: 1993.52
Fluctuation ratio: +0.00%
Traded volume:     363210
Traded price:      3394144

Curr date is 20130327
Last traded price: 1993.44
Fluctuation ratio: +0.49%
Traded volume:     346515
Traded price:      3793577

Curr date is 20130326
Last traded price: 1983.70
Fluctuation ratio: +0.30%
Traded volume:     287768
Traded price:      3423379

Curr date is 20130325
Last traded price: 1977.67
Fluctuation ratio: +1.49%
Traded volume:     271578
Traded price:      3407724

Curr date is 20130322
Last traded price: 1948.71
Fluctuation ratio: -0.11%
Traded volume:     251822
Traded price:      3302752

Curr date is 20130321
Last traded price: 1950.82
Fluctuation ratio: -0.44%
Traded volume:     257940
Traded price:      3488890

Curr date is 20130320
Last traded price: 1959.41
Fluctuation ratio: -0.97%
Traded volume:     326955
Traded price:      3727703

Curr date is 20130319
Last traded price: 1978.56
Fluctuation ratio: +0.53%
Traded volume:     322295
Traded price:      3646200

Curr date is 20130318
Last traded price: 1968.18
Fluctuation ratio: -0.92%
```

```
Traded volume:     300826
Traded price:      3906068

Curr date is 20130315
Last traded price: 1986.50
Fluctuation ratio: -0.78%
Traded volume:     303020
Traded price:      5211509

Curr date is 20130314
Last traded price: 2002.13
Fluctuation ratio: +0.12%
Traded volume:     321063
Traded price:      3971484

Curr date is 20130313
Last traded price: 1999.73
Fluctuation ratio: +0.32%
Traded volume:     349133
Traded price:      3420431

Curr date is 20130312
Last traded price: 1993.34
Fluctuation ratio: -0.50%
Traded volume:     386402
Traded price:      3281258

Curr date is 20130311
Last traded price: 2003.35
Fluctuation ratio: -0.13%
Traded volume:     394180
Traded price:      3759799

Curr date is 20130308
Last traded price: 2006.01
Fluctuation ratio: +0.08%
Traded volume:     307232
Traded price:      3514889

Curr date is 20130307
Last traded price: 2004.40
Fluctuation ratio: -0.81%
Traded volume:     382234
Traded price:      3639627

Curr date is 20130306
Last traded price: 2020.74
Fluctuation ratio: +0.20%
Traded volume:     391415
Traded price:      4044401

Curr date is 20130305
Last traded price: 2016.61
```

```
Fluctuation ratio: +0.17%
Traded volume:     487512
Traded price:      3726190

Curr date is 20130304
Last traded price: 2013.15
Fluctuation ratio: -0.66%
Traded volume:     351109
Traded price:      4142244

Curr date is 20130228
Last traded price: 2026.49
Fluctuation ratio: +1.12%
Traded volume:     416823
Traded price:      4098499

Curr date is 20130227
Last traded price: 2004.04
Fluctuation ratio: +0.20%
Traded volume:     412678
Traded price:      3465677

Curr date is 20130226
Last traded price: 2000.01
Fluctuation ratio: -0.47%
Traded volume:     404128
Traded price:      3369089

Curr date is 20130225
Last traded price: 2009.52
Fluctuation ratio: -0.46%
Traded volume:     333239
Traded price:      3092615

Curr date is 20130222
Last traded price: 2018.89
Fluctuation ratio: +0.18%
Traded volume:     402223
Traded price:      3954931

Curr date is 20130221
Last traded price: 2015.22
Fluctuation ratio: -0.47%
Traded volume:     386199
Traded price:      3959377

Curr date is 20130220
Last traded price: 2024.64
Fluctuation ratio: +1.95%
Traded volume:     402014
Traded price:      4824521

Curr date is 20130219
```

```
Last traded price: 1985.83
Fluctuation ratio: +0.20%
Traded volume:     340530
Traded price:      2756036

Curr date is 20130218
Last traded price: 1981.91
Fluctuation ratio: +0.04%
Traded volume:     412268
Traded price:      2808379

Curr date is 20130215
Last traded price: 1981.18
Fluctuation ratio: +0.08%
Traded volume:     575255
Traded price:      3219175

Curr date is 20130214
Last traded price: 1979.61
Fluctuation ratio: +0.18%
Traded volume:     330817
Traded price:      3354974

Curr date is 20130213
Last traded price: 1976.07
Fluctuation ratio: +1.56%
Traded volume:     325461
Traded price:      3617092

Curr date is 20130212
Last traded price: 1945.79
Fluctuation ratio: -0.26%
Traded volume:     303932
Traded price:      2805115

Curr date is 20130208
Last traded price: 1950.90
Fluctuation ratio: +0.99%
Traded volume:     286074
Traded price:      3843956

Curr date is 20130207
Last traded price: 1931.77
Fluctuation ratio: -0.23%
Traded volume:     407914
Traded price:      3510863

Curr date is 20130206
Last traded price: 1936.19
Fluctuation ratio: -0.10%
Traded volume:     344831
Traded price:      3599178
```

```
Curr date is 20130205
Last traded price: 1938.18
Fluctuation ratio: -0.77%
Traded volume:     362014
Traded price:      3703503

Curr date is 20130204
Last traded price: 1953.21
Fluctuation ratio: -0.23%
Traded volume:     340419
Traded price:      3495475

Curr date is 20130201
Last traded price: 1957.79
Fluctuation ratio: -0.21%
Traded volume:     378063
Traded price:      3955459

Curr date is 20130131
Last traded price: 1961.94
Fluctuation ratio: -0.13%
Traded volume:     394138
Traded price:      4366933

Curr date is 20130130
Last traded price: 1964.43
Fluctuation ratio: +0.43%
Traded volume:     516429
Traded price:      4207038

Curr date is 20130129
Last traded price: 1955.96
Fluctuation ratio: +0.84%
Traded volume:     459143
Traded price:      4537637

Curr date is 20130128
Last traded price: 1939.71
Fluctuation ratio: -0.36%
Traded volume:     343759
Traded price:      4528849

Curr date is 20130125
Last traded price: 1946.69
Fluctuation ratio: -0.91%
Traded volume:     395572
Traded price:      4955745

Curr date is 20130124
Last traded price: 1964.48
Fluctuation ratio: -0.80%
Traded volume:     551252
Traded price:      4449360
```

```
Curr date is 20130123
Last traded price: 1980.41
Fluctuation ratio: -0.81%
Traded volume:     485798
Traded price:      4125087

Curr date is 20130122
Last traded price: 1996.52
Fluctuation ratio: +0.49%
Traded volume:     548262
Traded price:      4180836

Curr date is 20130121
Last traded price: 1986.86
Fluctuation ratio: -0.05%
Traded volume:     440766
Traded price:      3751789

Curr date is 20130118
Last traded price: 1987.85
Fluctuation ratio: +0.69%
Traded volume:     576370
Traded price:      3830272

Curr date is 20130117
Last traded price: 1974.27
Fluctuation ratio: -0.16%
Traded volume:     805590
Traded price:      4031004

Curr date is 20130116
Last traded price: 1977.45
Fluctuation ratio: -0.32%
Traded volume:     453053
Traded price:      4227875

Curr date is 20130115
Last traded price: 1983.74
Fluctuation ratio: -1.16%
Traded volume:     542341
Traded price:      4376259

Curr date is 20130114
Last traded price: 2007.04
Fluctuation ratio: +0.52%
Traded volume:     449985
Traded price:      4280353

Curr date is 20130111
Last traded price: 1996.67
Fluctuation ratio: -0.50%
Traded volume:     467209
```

```
Traded price:      4363847

Curr date is 20130110
Last traded price: 2006.80
Fluctuation ratio: +0.75%
Traded volume:     494743
Traded price:      4471080

Curr date is 20130109
Last traded price: 1991.81
Fluctuation ratio: -0.31%
Traded volume:     597770
Traded price:      3943657

Curr date is 20130108
Last traded price: 1997.94
Fluctuation ratio: -0.66%
Traded volume:     489629
Traded price:      3904326

Curr date is 20130107
Last traded price: 2011.25
Fluctuation ratio: -0.03%
Traded volume:     532550
Traded price:      3941746

Curr date is 20130104
Last traded price: 2011.94
Fluctuation ratio: -0.37%
Traded volume:     372836
Traded price:      4676924

Curr date is 20130103
Last traded price: 2019.41
Fluctuation ratio: -0.58%
Traded volume:     485234
Traded price:      5987835

Curr date is 20130102
Last traded price: 2031.10
Fluctuation ratio: +1.71%
Traded volume:     359587
Traded price:      4400339

Curr date is 20121228
Last traded price: 1997.05
Fluctuation ratio: +0.49%
Traded volume:     290768
Traded price:      3188467

Curr date is 20121227
Last traded price: 1987.35
Fluctuation ratio: +0.26%
```

```
Traded volume:     372268
Traded price:      3015818

Curr date is 20121226
Last traded price: 1982.25
Fluctuation ratio: +0.02%
Traded volume:     420000
Traded price:      3473496

Curr date is 20121224
Last traded price: 1981.82
Fluctuation ratio: +0.07%
Traded volume:     343190
Traded price:      3181068

Curr date is 20121221
Last traded price: 1980.42
Fluctuation ratio: -0.95%
Traded volume:     544056
Traded price:      5166950

Curr date is 20121220
Last traded price: 1999.50
Fluctuation ratio: +0.32%
Traded volume:     416424
Traded price:      5237803

Curr date is 20121218
Last traded price: 1993.09
Fluctuation ratio: +0.51%
Traded volume:     447703
Traded price:      4557603

Curr date is 20121217
Last traded price: 1983.07
Fluctuation ratio: -0.60%
Traded volume:     361984
Traded price:      4204088

Curr date is 20121214
Last traded price: 1995.04
Fluctuation ratio: -0.39%
Traded volume:     345543
Traded price:      4250388

Curr date is 20121213
Last traded price: 2002.77
Fluctuation ratio: +1.38%
Traded volume:     414166
Traded price:      5893758

Curr date is 20121212
Last traded price: 1975.44
```

```
Fluctuation ratio: +0.55%
Traded volume:     453144
Traded price:      4561137

Curr date is 20121211
Last traded price: 1964.62
Fluctuation ratio: +0.37%
Traded volume:     399661
Traded price:      4427108

Curr date is 20121210
Last traded price: 1957.42
Fluctuation ratio: -0.00%
Traded volume:     361007
Traded price:      3607017

Curr date is 20121207
Last traded price: 1957.45
Fluctuation ratio: +0.40%
Traded volume:     405902
Traded price:      4246021

Curr date is 20121206
Last traded price: 1949.62
Fluctuation ratio: +0.13%
Traded volume:     335608
Traded price:      4597987

Curr date is 20121205
Last traded price: 1947.04
Fluctuation ratio: +0.61%
Traded volume:     483579
Traded price:      3993917

Curr date is 20121204
Last traded price: 1935.18
Fluctuation ratio: -0.25%
Traded volume:     288238
Traded price:      2799322

Curr date is 20121203
Last traded price: 1940.02
Fluctuation ratio: +0.37%
Traded volume:     303407
Traded price:      3112390

Curr date is 20121130
Last traded price: 1932.90
Fluctuation ratio: -0.10%
Traded volume:     331642
Traded price:      4432979

Curr date is 20121129
```

```
Last traded price: 1934.85
Fluctuation ratio: +1.15%
Traded volume:     382553
Traded price:      4758716

Curr date is 20121128
Last traded price: 1912.78
Fluctuation ratio: -0.65%
Traded volume:     403601
Traded price:      3835851

Curr date is 20121127
Last traded price: 1925.20
Fluctuation ratio: +0.87%
Traded volume:     581091
Traded price:      4675040

Curr date is 20121126
Last traded price: 1908.51
Fluctuation ratio: -0.15%
Traded volume:     273780
Traded price:      3178063

Curr date is 20121123
Last traded price: 1911.33
Fluctuation ratio: +0.62%
Traded volume:     324068
Traded price:      3536424

Curr date is 20121122
Last traded price: 1899.50
Fluctuation ratio: +0.82%
Traded volume:     407775
Traded price:      3690767

Curr date is 20121121
Last traded price: 1884.04
Fluctuation ratio: -0.32%
Traded volume:     371848
Traded price:      3894046

Curr date is 20121120
Last traded price: 1890.18
Fluctuation ratio: +0.64%
Traded volume:     368178
Traded price:      3662575

Curr date is 20121119
Last traded price: 1878.10
Fluctuation ratio: +0.93%
Traded volume:     346007
Traded price:      3204320
```

```
Curr date is 20121116
Last traded price: 1860.83
Fluctuation ratio: -0.53%
Traded volume:     459107
Traded price:      3987635

Curr date is 20121115
Last traded price: 1870.72
Fluctuation ratio: -1.23%
Traded volume:     446611
Traded price:      4136986

Curr date is 20121114
Last traded price: 1894.04
Fluctuation ratio: +0.23%
Traded volume:     525579
Traded price:      4698989

Curr date is 20121113
Last traded price: 1889.70
Fluctuation ratio: -0.59%
Traded volume:     487154
Traded price:      3859300

Curr date is 20121112
Last traded price: 1900.87
Fluctuation ratio: -0.19%
Traded volume:     468990
Traded price:      3735229

Curr date is 20121109
Last traded price: 1904.41
Fluctuation ratio: -0.52%
Traded volume:     628749
Traded price:      4745456

Curr date is 20121108
Last traded price: 1914.41
Fluctuation ratio: -1.19%
Traded volume:     653959
Traded price:      5303730

Curr date is 20121107
Last traded price: 1937.55
Fluctuation ratio: +0.49%
Traded volume:     550293
Traded price:      4579636

Curr date is 20121106
Last traded price: 1928.17
Fluctuation ratio: +1.05%
Traded volume:     539961
Traded price:      4500836
```

```
Curr date is 20121105
Last traded price: 1908.22
Fluctuation ratio: -0.55%
Traded volume:     336715
Traded price:      4538241

Curr date is 20121102
Last traded price: 1918.72
Fluctuation ratio: +1.07%
Traded volume:     424542
Traded price:      5086355

Curr date is 20121101
Last traded price: 1898.44
Fluctuation ratio: -0.71%
Traded volume:     396216
Traded price:      5177401

Curr date is 20121031
Last traded price: 1912.06
Fluctuation ratio: +0.66%
Traded volume:     392656
Traded price:      3751541

Curr date is 20121030
Last traded price: 1899.58
Fluctuation ratio: +0.43%
Traded volume:     326074
Traded price:      3388630

Curr date is 20121029
Last traded price: 1891.52
Fluctuation ratio: +0.00%
Traded volume:     366421
Traded price:      4120316

Curr date is 20121026
Last traded price: 1891.43
Fluctuation ratio: -1.72%
Traded volume:     432671
Traded price:      4551182

Curr date is 20121025
Last traded price: 1924.50
Fluctuation ratio: +0.55%
Traded volume:     430280
Traded price:      4769602

Curr date is 20121024
Last traded price: 1913.96
Fluctuation ratio: -0.67%
Traded volume:     558527
```

```
Traded price:      5569470

Curr date is 20121023
Last traded price: 1926.81
Fluctuation ratio: -0.76%
Traded volume:     528610
Traded price:      4102102

Curr date is 20121022
Last traded price: 1941.59
Fluctuation ratio: -0.12%
Traded volume:     454125
Traded price:      3767981

Curr date is 20121019
Last traded price: 1943.84
Fluctuation ratio: -0.78%
Traded volume:     377677
Traded price:      4056203

Curr date is 20121018
Last traded price: 1959.12
Fluctuation ratio: +0.20%
Traded volume:     490221
Traded price:      4212529

Curr date is 20121017
Last traded price: 1955.15
Fluctuation ratio: +0.70%
Traded volume:     494091
Traded price:      4143505

Curr date is 20121016
Last traded price: 1941.54
Fluctuation ratio: +0.83%
Traded volume:     527247
Traded price:      3775270

Curr date is 20121015
Last traded price: 1925.59
Fluctuation ratio: -0.40%
Traded volume:     555239
Traded price:      4488250

Curr date is 20121012
Last traded price: 1933.26
Fluctuation ratio: +0.01%
Traded volume:     389850
Traded price:      3718129

Curr date is 20121011
Last traded price: 1933.09
Fluctuation ratio: -0.78%
```

```
Traded volume:     437554
Traded price:      4491546

Curr date is 20121010
Last traded price: 1948.22
Fluctuation ratio: -1.56%
Traded volume:     419284
Traded price:      4033853

Curr date is 20121009
Last traded price: 1979.04
Fluctuation ratio: -0.14%
Traded volume:     510522
Traded price:      3729371

Curr date is 20121008
Last traded price: 1981.89
Fluctuation ratio: -0.67%
Traded volume:     497669
Traded price:      4020885

Curr date is 20121005
Last traded price: 1995.17
Fluctuation ratio: +0.12%
Traded volume:     672497
Traded price:      4399645

Curr date is 20121004
Last traded price: 1992.68
Fluctuation ratio: -0.17%
Traded volume:     584880
Traded price:      4819806

Curr date is 20121002
Last traded price: 1996.03
Fluctuation ratio: -0.01%
Traded volume:     491911
Traded price:      3603407

Curr date is 20120928
Last traded price: 1996.21
Fluctuation ratio: +0.38%
Traded volume:     667396
Traded price:      4713090

Curr date is 20120927
Last traded price: 1988.70
Fluctuation ratio: +0.42%
Traded volume:     686814
Traded price:      4656787

Curr date is 20120926
Last traded price: 1980.44
```

```
Fluctuation ratio: -0.55%
Traded volume:     1190145
Traded price:      4730197

Curr date is 20120925
Last traded price: 1991.41
Fluctuation ratio: -0.60%
Traded volume:     523722
Traded price:      4452530

Curr date is 20120924
Last traded price: 2003.44
Fluctuation ratio: +0.05%
Traded volume:     476877
Traded price:      4188887

Curr date is 20120921
Last traded price: 2002.37
Fluctuation ratio: +0.60%
Traded volume:     642062
Traded price:      4904097

Curr date is 20120920
Last traded price: 1990.33
Fluctuation ratio: -0.87%
Traded volume:     612526
Traded price:      5275669

Curr date is 20120919
Last traded price: 2007.88
Fluctuation ratio: +0.15%
Traded volume:     945798
Traded price:      5702726

Curr date is 20120918
Last traded price: 2004.96
Fluctuation ratio: +0.13%
Traded volume:     845335
Traded price:      5032350

Curr date is 20120917
Last traded price: 2002.35
Fluctuation ratio: -0.26%
Traded volume:     871250
Traded price:      6502601

Curr date is 20120914
Last traded price: 2007.58
Fluctuation ratio: +2.92%
Traded volume:     1012996
Traded price:      9071183

Curr date is 20120913
```

```
Last traded price: 1950.69
Fluctuation ratio: +0.03%
Traded volume:     968994
Traded price:      5968682

Curr date is 20120912
Last traded price: 1950.03
Fluctuation ratio: +1.56%
Traded volume:     699616
Traded price:      5150979

Curr date is 20120911
Last traded price: 1920.00
Fluctuation ratio: -0.24%
Traded volume:     707288
Traded price:      4141528

Curr date is 20120910
Last traded price: 1924.70
Fluctuation ratio: -0.25%
Traded volume:     614131
Traded price:      4277687

Curr date is 20120907
Last traded price: 1929.58
Fluctuation ratio: +2.57%
Traded volume:     588839
Traded price:      5754333

Curr date is 20120906
Last traded price: 1881.24
Fluctuation ratio: +0.38%
Traded volume:     740858
Traded price:      4167135

Curr date is 20120905
Last traded price: 1874.03
Fluctuation ratio: -1.74%
Traded volume:     615132
Traded price:      4622580

Curr date is 20120904
Last traded price: 1907.13
Fluctuation ratio: -0.29%
Traded volume:     868713
Traded price:      4512475

Curr date is 20120903
Last traded price: 1912.71
Fluctuation ratio: +0.40%
Traded volume:     587695
Traded price:      4176233
```

```
Curr date is 20120831
Last traded price: 1905.12
Fluctuation ratio: -0.07%
Traded volume:     639324
Traded price:      4394387

Curr date is 20120830
Last traded price: 1906.38
Fluctuation ratio: -1.15%
Traded volume:     714998
Traded price:      4847192

Curr date is 20120829
Last traded price: 1928.54
Fluctuation ratio: +0.64%
Traded volume:     578853
Traded price:      4766824

Curr date is 20120828
Last traded price: 1916.33
Fluctuation ratio: -0.08%
Traded volume:     601127
Traded price:      4358101

Curr date is 20120827
Last traded price: 1917.87
Fluctuation ratio: -0.10%
Traded volume:     563662
Traded price:      5961544

Curr date is 20120824
Last traded price: 1919.81
Fluctuation ratio: -1.17%
Traded volume:     615571
Traded price:      4278633

Curr date is 20120823
Last traded price: 1942.54
Fluctuation ratio: +0.38%
Traded volume:     594864
Traded price:      4133832

Curr date is 20120822
Last traded price: 1935.19
Fluctuation ratio: -0.41%
Traded volume:     417708
Traded price:      4016170

Curr date is 20120821
Last traded price: 1943.22
Fluctuation ratio: -0.16%
Traded volume:     402268
Traded price:      3723781
```

```
Curr date is 20120820
Last traded price: 1946.31
Fluctuation ratio: -0.01%
Traded volume:     353453
Traded price:      3786672

Curr date is 20120817
Last traded price: 1946.54
Fluctuation ratio: -0.58%
Traded volume:     366494
Traded price:      4449859

Curr date is 20120816
Last traded price: 1957.91
Fluctuation ratio: +0.05%
Traded volume:     426141
Traded price:      3972935

Curr date is 20120814
Last traded price: 1956.96
Fluctuation ratio: +1.27%
Traded volume:     336639
Traded price:      4005981

Curr date is 20120813
Last traded price: 1932.44
Fluctuation ratio: -0.72%
Traded volume:     348150
Traded price:      3114783

Curr date is 20120810
Last traded price: 1946.40
Fluctuation ratio: +0.30%
Traded volume:     320982
Traded price:      4000111

Curr date is 20120809
Last traded price: 1940.59
Fluctuation ratio: +1.96%
Traded volume:     487612
Traded price:      6426153

Curr date is 20120808
Last traded price: 1903.23
Fluctuation ratio: +0.87%
Traded volume:     467564
Traded price:      5147435

Curr date is 20120807
Last traded price: 1886.80
Fluctuation ratio: +0.05%
Traded volume:     320454
```

```
Traded price:      3465101

Curr date is 20120806
Last traded price: 1885.88
Fluctuation ratio: +2.01%
Traded volume:     242753
Traded price:      4127403

Curr date is 20120803
Last traded price: 1848.68
Fluctuation ratio: -1.11%
Traded volume:     254463
Traded price:      3601771

Curr date is 20120802
Last traded price: 1869.40
Fluctuation ratio: -0.56%
Traded volume:     267526
Traded price:      3350369

Curr date is 20120801
Last traded price: 1879.93
Fluctuation ratio: -0.11%
Traded volume:     285300
Traded price:      3733696

Curr date is 20120731
Last traded price: 1881.99
Fluctuation ratio: +2.07%
Traded volume:     353945
Traded price:      5535572

Curr date is 20120730
Last traded price: 1843.79
Fluctuation ratio: +0.80%
Traded volume:     333494
Traded price:      3833265

Curr date is 20120727
Last traded price: 1829.16
Fluctuation ratio: +2.62%
Traded volume:     394116
Traded price:      4543305

Curr date is 20120726
Last traded price: 1782.47
Fluctuation ratio: +0.74%
Traded volume:     404503
Traded price:      3766267

Curr date is 20120725
Last traded price: 1769.31
Fluctuation ratio: -1.37%
```

```
Traded volume:     460413
Traded price:      3771825

Curr date is 20120724
Last traded price: 1793.93
Fluctuation ratio: +0.25%
Traded volume:     480765
Traded price:      4074951

Curr date is 20120723
Last traded price: 1789.44
Fluctuation ratio: -1.84%
Traded volume:     494653
Traded price:      3588133

Curr date is 20120720
Last traded price: 1822.93
Fluctuation ratio: -0.00%
Traded volume:     405415
Traded price:      3515080

Curr date is 20120719
Last traded price: 1822.96
Fluctuation ratio: +1.56%
Traded volume:     482364
Traded price:      4624045

Curr date is 20120718
Last traded price: 1794.91
Fluctuation ratio: -1.48%
Traded volume:     448440
Traded price:      4366435

Curr date is 20120717
Last traded price: 1821.96
Fluctuation ratio: +0.23%
Traded volume:     401950
Traded price:      4025861

Curr date is 20120716
Last traded price: 1817.79
Fluctuation ratio: +0.27%
Traded volume:     263491
Traded price:      3186678

Curr date is 20120713
Last traded price: 1812.89
Fluctuation ratio: +1.54%
Traded volume:     344309
Traded price:      4663049

Curr date is 20120712
Last traded price: 1785.39
```

```
Fluctuation ratio: -2.24%
Traded volume:     314318
Traded price:      4213460

Curr date is 20120711
Last traded price: 1826.39
Fluctuation ratio: -0.17%
Traded volume:     332443
Traded price:      3285617

Curr date is 20120710
Last traded price: 1829.45
Fluctuation ratio: -0.36%
Traded volume:     363939
Traded price:      3314026

Curr date is 20120709
Last traded price: 1836.13
Fluctuation ratio: -1.19%
Traded volume:     277613
Traded price:      3252325

Curr date is 20120706
Last traded price: 1858.20
Fluctuation ratio: -0.92%
Traded volume:     335727
Traded price:      3559298

Curr date is 20120705
Last traded price: 1875.49
Fluctuation ratio: +0.06%
Traded volume:     317750
Traded price:      3543395

Curr date is 20120704
Last traded price: 1874.45
Fluctuation ratio: +0.35%
Traded volume:     332982
Traded price:      4532135

Curr date is 20120703
Last traded price: 1867.82
Fluctuation ratio: +0.87%
Traded volume:     344449
Traded price:      4772351

Curr date is 20120702
Last traded price: 1851.65
Fluctuation ratio: -0.13%
Traded volume:     270640
Traded price:      3542724

Curr date is 20120629
```

```
Last traded price: 1854.01
Fluctuation ratio: +1.91%
Traded volume:     309937
Traded price:      4293002

Curr date is 20120628
Last traded price: 1819.18
Fluctuation ratio: +0.08%
Traded volume:     305118
Traded price:      3090506

Curr date is 20120627
Last traded price: 1817.65
Fluctuation ratio: -0.01%
Traded volume:     314211
Traded price:      4232361

Curr date is 20120626
Last traded price: 1817.81
Fluctuation ratio: -0.41%
Traded volume:     307708
Traded price:      3581421

Curr date is 20120625
Last traded price: 1825.38
Fluctuation ratio: -1.19%
Traded volume:     286909
Traded price:      4078253

Curr date is 20120622
Last traded price: 1847.39
Fluctuation ratio: -2.21%
Traded volume:     295808
Traded price:      4280856

Curr date is 20120621
Last traded price: 1889.15
Fluctuation ratio: -0.79%
Traded volume:     383877
Traded price:      3735977

Curr date is 20120620
Last traded price: 1904.12
Fluctuation ratio: +0.65%
Traded volume:     362348
Traded price:      3635005

Curr date is 20120619
Last traded price: 1891.77
Fluctuation ratio: +0.00%
Traded volume:     376224
Traded price:      3443467
```

```
Curr date is 20120618
Last traded price: 1891.71
Fluctuation ratio: +1.81%
Traded volume:     371549
Traded price:      4386792

Curr date is 20120615
Last traded price: 1858.16
Fluctuation ratio: -0.71%
Traded volume:     436672
Traded price:      4449366

Curr date is 20120614
Last traded price: 1871.48
Fluctuation ratio: +0.65%
Traded volume:     419392
Traded price:      4765353

Curr date is 20120613
Last traded price: 1859.32
Fluctuation ratio: +0.25%
Traded volume:     549884
Traded price:      3735364

Curr date is 20120612
Last traded price: 1854.74
Fluctuation ratio: -0.66%
Traded volume:     417375
Traded price:      3587653

Curr date is 20120611
Last traded price: 1867.04
Fluctuation ratio: +1.71%
Traded volume:     371237
Traded price:      4226281

Curr date is 20120608
Last traded price: 1835.64
Fluctuation ratio: -0.67%
Traded volume:     373822
Traded price:      3397742

Curr date is 20120607
Last traded price: 1847.95
Fluctuation ratio: +2.56%
Traded volume:     391582
Traded price:      4524807

Curr date is 20120605
Last traded price: 1801.85
Fluctuation ratio: +1.05%
Traded volume:     331920
Traded price:      3475416
```

```
Curr date is 20120604
Last traded price: 1783.13
Fluctuation ratio: -2.80%
Traded volume:     387025
Traded price:      4859518

Curr date is 20120601
Last traded price: 1834.51
Fluctuation ratio: -0.49%
Traded volume:     328985
Traded price:      3472674

Curr date is 20120531
Last traded price: 1843.47
Fluctuation ratio: -0.08%
Traded volume:     442179
Traded price:      4563692

Curr date is 20120530
Last traded price: 1844.86
Fluctuation ratio: -0.27%
Traded volume:     399300
Traded price:      3621755

Curr date is 20120529
Last traded price: 1849.91
Fluctuation ratio: +1.41%
Traded volume:     321007
Traded price:      3698861

Curr date is 20120525
Last traded price: 1824.17
Fluctuation ratio: +0.53%
Traded volume:     380520
Traded price:      3485663

Curr date is 20120524
Last traded price: 1814.47
Fluctuation ratio: +0.32%
Traded volume:     401671
Traded price:      3481859

Curr date is 20120523
Last traded price: 1808.62
Fluctuation ratio: -1.10%
Traded volume:     440968
Traded price:      4149950

Curr date is 20120522
Last traded price: 1828.69
Fluctuation ratio: +1.64%
Traded volume:     418659
```

```
Traded price:      4799589

Curr date is 20120521
Last traded price: 1799.13
Fluctuation ratio: +0.94%
Traded volume:     408565
Traded price:      4214282

Curr date is 20120518
Last traded price: 1782.46
Fluctuation ratio: -3.40%
Traded volume:     539564
Traded price:      5654878

Curr date is 20120517
Last traded price: 1845.24
Fluctuation ratio: +0.26%
Traded volume:     435950
Traded price:      5589086

Curr date is 20120516
Last traded price: 1840.53
Fluctuation ratio: -3.08%
Traded volume:     419869
Traded price:      5468681

Curr date is 20120515
Last traded price: 1898.96
Fluctuation ratio: -0.77%
Traded volume:     516768
Traded price:      5114692

Curr date is 20120514
Last traded price: 1913.73
Fluctuation ratio: -0.18%
Traded volume:     391974
Traded price:      3964754

Curr date is 20120511
Last traded price: 1917.13
Fluctuation ratio: -1.43%
Traded volume:     529990
Traded price:      4334830

Curr date is 20120510
Last traded price: 1944.93
Fluctuation ratio: -0.27%
Traded volume:     418439
Traded price:      4379618

Curr date is 20120509
Last traded price: 1950.29
Fluctuation ratio: -0.85%
```

```
Traded volume:     462645
Traded price:      4893642

Curr date is 20120508
Last traded price: 1967.01
Fluctuation ratio: +0.54%
Traded volume:     486227
Traded price:      4107902

Curr date is 20120507
Last traded price: 1956.44
Fluctuation ratio: -1.64%
Traded volume:     528564
Traded price:      4703551

Curr date is 20120504
Last traded price: 1989.15
Fluctuation ratio: -0.30%
Traded volume:     601911
Traded price:      4828699

Curr date is 20120503
Last traded price: 1995.11
Fluctuation ratio: -0.20%
Traded volume:     676521
Traded price:      4572074

Curr date is 20120502
Last traded price: 1999.07
Fluctuation ratio: +0.86%
Traded volume:     643613
Traded price:      6227639

Curr date is 20120430
Last traded price: 1981.99
Fluctuation ratio: +0.34%
Traded volume:     828566
Traded price:      6235246

Curr date is 20120427
Last traded price: 1975.35
Fluctuation ratio: +0.58%
Traded volume:     474099
Traded price:      5284923

Curr date is 20120426
Last traded price: 1964.04
Fluctuation ratio: +0.10%
Traded volume:     599192
Traded price:      5268794

Curr date is 20120425
Last traded price: 1961.98
```

```
Fluctuation ratio: -0.07%
Traded volume:     674156
Traded price:      4941163

Curr date is 20120424
Last traded price: 1963.42
Fluctuation ratio: -0.47%
Traded volume:     624583
Traded price:      4983570

Curr date is 20120423
Last traded price: 1972.63
Fluctuation ratio: -0.10%
Traded volume:     578388
Traded price:      3520554

Curr date is 20120420
Last traded price: 1974.65
Fluctuation ratio: -1.26%
Traded volume:     511708
Traded price:      4750644

Curr date is 20120419
Last traded price: 1999.86
Fluctuation ratio: -0.23%
Traded volume:     406776
Traded price:      4030917

Curr date is 20120418
Last traded price: 2004.53
Fluctuation ratio: +0.97%
Traded volume:     378419
Traded price:      4566362

Curr date is 20120417
Last traded price: 1985.30
Fluctuation ratio: -0.37%
Traded volume:     441215
Traded price:      3931365

Curr date is 20120416
Last traded price: 1992.63
Fluctuation ratio: -0.81%
Traded volume:     447291
Traded price:      3846819

Curr date is 20120413
Last traded price: 2008.91
Fluctuation ratio: +1.12%
Traded volume:     488834
Traded price:      5082726

Curr date is 20120412
```

```
Last traded price: 1986.63
Fluctuation ratio: -0.39%
Traded volume:     387132
Traded price:      4921483

Curr date is 20120410
Last traded price: 1994.41
Fluctuation ratio: -0.13%
Traded volume:     536172
Traded price:      4826007

Curr date is 20120409
Last traded price: 1997.08
Fluctuation ratio: -1.57%
Traded volume:     462475
Traded price:      4275840

Curr date is 20120406
Last traded price: 2029.03
Fluctuation ratio: +0.01%
Traded volume:     529233
Traded price:      4391276

Curr date is 20120405
Last traded price: 2028.77
Fluctuation ratio: +0.50%
Traded volume:     541828
Traded price:      5642717

Curr date is 20120404
Last traded price: 2018.61
Fluctuation ratio: -1.50%
Traded volume:     607177
Traded price:      5766833

Curr date is 20120403
Last traded price: 2049.28
Fluctuation ratio: +0.99%
Traded volume:     587553
Traded price:      5986222

Curr date is 20120402
Last traded price: 2029.29
Fluctuation ratio: +0.76%
Traded volume:     405802
Traded price:      4280405

Curr date is 20120330
Last traded price: 2014.04
Fluctuation ratio: -0.02%
Traded volume:     445203
Traded price:      5247702
```

```
Curr date is 20120329
Last traded price: 2014.41
Fluctuation ratio: -0.85%
Traded volume:     445611
Traded price:      4906232

Curr date is 20120328
Last traded price: 2031.74
Fluctuation ratio: -0.39%
Traded volume:     526414
Traded price:      4994050

Curr date is 20120327
Last traded price: 2039.76
Fluctuation ratio: +1.02%
Traded volume:     599121
Traded price:      5076394

Curr date is 20120326
Last traded price: 2019.19
Fluctuation ratio: -0.38%
Traded volume:     431293
Traded price:      4512679

Curr date is 20120323
Last traded price: 2026.83
Fluctuation ratio: +0.04%
Traded volume:     501021
Traded price:      4763755

Curr date is 20120322
Last traded price: 2026.12
Fluctuation ratio: -0.05%
Traded volume:     535735
Traded price:      5194740

Curr date is 20120321
Last traded price: 2027.23
Fluctuation ratio: -0.73%
Traded volume:     511340
Traded price:      5083846

Curr date is 20120320
Last traded price: 2042.15
Fluctuation ratio: -0.24%
Traded volume:     535437
Traded price:      5331966

Curr date is 20120319
Last traded price: 2047.00
Fluctuation ratio: +0.62%
Traded volume:     545600
Traded price:      4866743
```

```
Curr date is 20120316
Last traded price: 2034.44
Fluctuation ratio: -0.46%
Traded volume:     441029
Traded price:      5093085

Curr date is 20120315
Last traded price: 2043.76
Fluctuation ratio: -0.06%
Traded volume:     470866
Traded price:      5912970

Curr date is 20120314
Last traded price: 2045.08
Fluctuation ratio: +0.99%
Traded volume:     534813
Traded price:      6310496

Curr date is 20120313
Last traded price: 2025.04
Fluctuation ratio: +1.13%
Traded volume:     443360
Traded price:      5582396

Curr date is 20120312
Last traded price: 2002.50
Fluctuation ratio: -0.78%
Traded volume:     369720
Traded price:      4733609

Curr date is 20120309
Last traded price: 2018.30
Fluctuation ratio: +0.88%
Traded volume:     407404
Traded price:      5296148

Curr date is 20120308
Last traded price: 2000.76
Fluctuation ratio: +0.94%
Traded volume:     450439
Traded price:      6243126

Curr date is 20120307
Last traded price: 1982.15
Fluctuation ratio: -0.91%
Traded volume:     506010
Traded price:      4908154

Curr date is 20120306
Last traded price: 2000.36
Fluctuation ratio: -0.78%
Traded volume:     549184
```

```
Traded price:      5234313

Curr date is 20120305
Last traded price: 2016.06
Fluctuation ratio: -0.91%
Traded volume:     542267
Traded price:      5058292

Curr date is 20120302
Last traded price: 2034.63
Fluctuation ratio: +0.22%
Traded volume:     476623
Traded price:      5761444

Curr date is 20120229
Last traded price: 2030.25
Fluctuation ratio: +1.33%
Traded volume:     554932
Traded price:      6708784

Curr date is 20120228
Last traded price: 2003.69
Fluctuation ratio: +0.63%
Traded volume:     599312
Traded price:      6255899

Curr date is 20120227
Last traded price: 1991.16
Fluctuation ratio: -1.42%
Traded volume:     629045
Traded price:      5479357

Curr date is 20120224
Last traded price: 2019.89
Fluctuation ratio: +0.60%
Traded volume:     709449
Traded price:      5630632

Curr date is 20120223
Last traded price: 2007.80
Fluctuation ratio: -1.03%
Traded volume:     679397
Traded price:      6270656

Curr date is 20120222
Last traded price: 2028.65
Fluctuation ratio: +0.22%
Traded volume:     874898
Traded price:      6104277

Curr date is 20120221
Last traded price: 2024.24
Fluctuation ratio: -0.03%
```

```
Traded volume:     701906
Traded price:      5551942

Curr date is 20120220
Last traded price: 2024.90
Fluctuation ratio: +0.07%
Traded volume:     723783
Traded price:      5954187

Curr date is 20120217
Last traded price: 2023.47
Fluctuation ratio: +1.30%
Traded volume:     724644
Traded price:      7387348

Curr date is 20120216
Last traded price: 1997.45
Fluctuation ratio: -1.38%
Traded volume:     690390
Traded price:      6768121

Curr date is 20120215
Last traded price: 2025.32
Fluctuation ratio: +1.13%
Traded volume:     675963
Traded price:      7604871

Curr date is 20120214
Last traded price: 2002.64
Fluctuation ratio: -0.15%
Traded volume:     500085
Traded price:      6216305

Curr date is 20120213
Last traded price: 2005.74
Fluctuation ratio: +0.60%
Traded volume:     448349
Traded price:      5676017

Curr date is 20120210
Last traded price: 1993.71
Fluctuation ratio: -1.04%
Traded volume:     689915
Traded price:      7499417

Curr date is 20120209
Last traded price: 2014.62
Fluctuation ratio: +0.54%
Traded volume:     551411
Traded price:      7865691

Curr date is 20120208
Last traded price: 2003.73
```

```
Fluctuation ratio: +1.12%
Traded volume:     582088
Traded price:      7379314

Curr date is 20120207
Last traded price: 1981.59
Fluctuation ratio: +0.43%
Traded volume:     533879
Traded price:      6501793

Curr date is 20120206
Last traded price: 1973.13
Fluctuation ratio: +0.04%
Traded volume:     517914
Traded price:      7225740

Curr date is 20120203
Last traded price: 1972.34
Fluctuation ratio: -0.60%
Traded volume:     530954
Traded price:      7418238

Curr date is 20120202
Last traded price: 1984.30
Fluctuation ratio: +1.28%
Traded volume:     628953
Traded price:      8594563

Curr date is 20120201
Last traded price: 1959.24
Fluctuation ratio: +0.18%
Traded volume:     494655
Traded price:      7036627

Curr date is 20120131
Last traded price: 1955.79
Fluctuation ratio: +0.79%
Traded volume:     464111
Traded price:      6584883

Curr date is 20120130
Last traded price: 1940.55
Fluctuation ratio: -1.24%
Traded volume:     407617
Traded price:      6327041

Curr date is 20120127
Last traded price: 1964.83
Fluctuation ratio: +0.39%
Traded volume:     443126
Traded price:      6022669

Curr date is 20120126
```

```
Last traded price: 1957.18
Fluctuation ratio: +0.25%
Traded volume:     403186
Traded price:      5458618

Curr date is 20120125
Last traded price: 1952.23
Fluctuation ratio: +0.12%
Traded volume:     371819
Traded price:      6064992

Curr date is 20120120
Last traded price: 1949.89
Fluctuation ratio: +1.82%
Traded volume:     397904
Traded price:      7239003

Curr date is 20120119
Last traded price: 1914.97
Fluctuation ratio: +1.19%
Traded volume:     368905
Traded price:      6345480

Curr date is 20120118
Last traded price: 1892.39
Fluctuation ratio: -0.02%
Traded volume:     422588
Traded price:      5952350

Curr date is 20120117
Last traded price: 1892.74
Fluctuation ratio: +1.80%
Traded volume:     426546
Traded price:      5873748

Curr date is 20120116
Last traded price: 1859.27
Fluctuation ratio: -0.87%
Traded volume:     383100
Traded price:      3864750

Curr date is 20120113
Last traded price: 1875.68
Fluctuation ratio: +0.60%
Traded volume:     465975
Traded price:      5170645

Curr date is 20120112
Last traded price: 1864.57
Fluctuation ratio: +1.03%
Traded volume:     471019
Traded price:      5309593
```

```
Curr date is 20120111
Last traded price: 1845.55
Fluctuation ratio: -0.41%
Traded volume:     442470
Traded price:      4762020

Curr date is 20120110
Last traded price: 1853.22
Fluctuation ratio: +1.46%
Traded volume:     383089
Traded price:      4772192

Curr date is 20120109
Last traded price: 1826.49
Fluctuation ratio: -0.90%
Traded volume:     388572
Traded price:      4134305

Curr date is 20120106
Last traded price: 1843.14
Fluctuation ratio: -1.11%
Traded volume:     466882
Traded price:      4356678

Curr date is 20120105
Last traded price: 1863.74
Fluctuation ratio: -0.13%
Traded volume:     528824
Traded price:      4793705

Curr date is 20120104
Last traded price: 1866.22
Fluctuation ratio: -0.49%
Traded volume:     488419
Traded price:      4655044

Curr date is 20120103
Last traded price: 1875.41
Fluctuation ratio: +2.69%
Traded volume:     438483
Traded price:      4785291

Curr date is 20120102
Last traded price: 1826.37
Fluctuation ratio: +0.03%
Traded volume:     318982
Traded price:      3231265

Curr date is 20111229
Last traded price: 1825.74
Fluctuation ratio: +0.03%
Traded volume:     335707
Traded price:      3365860
```

```
Curr date is 20111228
Last traded price: 1825.12
Fluctuation ratio: -0.92%
Traded volume:     374549
Traded price:      3260370

Curr date is 20111227
Last traded price: 1842.02
Fluctuation ratio: -0.79%
Traded volume:     521434
Traded price:      4182552

Curr date is 20111226
Last traded price: 1856.70
Fluctuation ratio: -0.56%
Traded volume:     425108
Traded price:      2972128

Curr date is 20111223
Last traded price: 1867.22
Fluctuation ratio: +1.07%
Traded volume:     458481
Traded price:      4209943

Curr date is 20111222
Last traded price: 1847.49
Fluctuation ratio: -0.05%
Traded volume:     400464
Traded price:      3071394

Curr date is 20111221
Last traded price: 1848.41
Fluctuation ratio: +3.09%
Traded volume:     509547
Traded price:      4887915

Curr date is 20111220
Last traded price: 1793.06
Fluctuation ratio: +0.91%
Traded volume:     385665
Traded price:      3913314

Curr date is 20111219
Last traded price: 1776.93
Fluctuation ratio: -3.43%
Traded volume:     676680
Traded price:      6581501

Curr date is 20111216
Last traded price: 1839.96
Fluctuation ratio: +1.15%
Traded volume:     484186
```

```
Traded price:      4292758

Curr date is 20111215
Last traded price: 1819.11
Fluctuation ratio: -2.08%
Traded volume:     403326
Traded price:      5319597

Curr date is 20111214
Last traded price: 1857.75
Fluctuation ratio: -0.34%
Traded volume:     397193
Traded price:      4355636

Curr date is 20111213
Last traded price: 1864.06
Fluctuation ratio: -1.88%
Traded volume:     444585
Traded price:      5210178

Curr date is 20111212
Last traded price: 1899.76
Fluctuation ratio: +1.33%
Traded volume:     405097
Traded price:      5303702

Curr date is 20111209
Last traded price: 1874.75
Fluctuation ratio: -1.97%
Traded volume:     367333
Traded price:      5025076

Curr date is 20111208
Last traded price: 1912.39
Fluctuation ratio: -0.37%
Traded volume:     420434
Traded price:      5849587

Curr date is 20111207
Last traded price: 1919.42
Fluctuation ratio: +0.87%
Traded volume:     396686
Traded price:      5691433

Curr date is 20111206
Last traded price: 1902.82
Fluctuation ratio: -1.04%
Traded volume:     379676
Traded price:      4562619

Curr date is 20111205
Last traded price: 1922.90
Fluctuation ratio: +0.36%
```

```
Traded volume:     325509
Traded price:      4094778

Curr date is 20111202
Last traded price: 1916.04
Fluctuation ratio: -0.01%
Traded volume:     380509
Traded price:      5303675

Curr date is 20111201
Last traded price: 1916.18
Fluctuation ratio: +3.72%
Traded volume:     495785
Traded price:      9232409

Curr date is 20111130
Last traded price: 1847.51
Fluctuation ratio: -0.49%
Traded volume:     327558
Traded price:      5994526

Curr date is 20111129
Last traded price: 1856.52
Fluctuation ratio: +2.27%
Traded volume:     324289
Traded price:      6119167

Curr date is 20111128
Last traded price: 1815.28
Fluctuation ratio: +2.19%
Traded volume:     272246
Traded price:      4195427

Curr date is 20111125
Last traded price: 1776.40
Fluctuation ratio: -1.04%
Traded volume:     301196
Traded price:      4158716

Curr date is 20111124
Last traded price: 1795.06
Fluctuation ratio: +0.67%
Traded volume:     275641
Traded price:      4080949

Curr date is 20111123
Last traded price: 1783.10
Fluctuation ratio: -2.36%
Traded volume:     406817
Traded price:      4592353

Curr date is 20111122
Last traded price: 1826.28
```

```
Fluctuation ratio: +0.34%
Traded volume:     261447
Traded price:      3818063

Curr date is 20111121
Last traded price: 1820.03
Fluctuation ratio: -1.04%
Traded volume:     296574
Traded price:      3925704

Curr date is 20111118
Last traded price: 1839.17
Fluctuation ratio: -2.00%
Traded volume:     327940
Traded price:      4712681

Curr date is 20111117
Last traded price: 1876.67
Fluctuation ratio: +1.11%
Traded volume:     348733
Traded price:      4666947

Curr date is 20111116
Last traded price: 1856.07
Fluctuation ratio: -1.59%
Traded volume:     402001
Traded price:      5579052

Curr date is 20111115
Last traded price: 1886.12
Fluctuation ratio: -0.88%
Traded volume:     355841
Traded price:      4738345

Curr date is 20111114
Last traded price: 1902.81
Fluctuation ratio: +2.11%
Traded volume:     323066
Traded price:      5185948

Curr date is 20111111
Last traded price: 1863.45
Fluctuation ratio: +2.77%
Traded volume:     291710
Traded price:      5147866

Curr date is 20111110
Last traded price: 1813.25
Fluctuation ratio: -4.94%
Traded volume:     391746
Traded price:      7677643

Curr date is 20111109
```

```
Last traded price: 1907.53
Fluctuation ratio: +0.23%
Traded volume:     414346
Traded price:      5474561

Curr date is 20111108
Last traded price: 1903.14
Fluctuation ratio: -0.83%
Traded volume:     398620
Traded price:      5156384

Curr date is 20111107
Last traded price: 1919.10
Fluctuation ratio: -0.48%
Traded volume:     342958
Traded price:      5071524

Curr date is 20111104
Last traded price: 1928.41
Fluctuation ratio: +3.13%
Traded volume:     461468
Traded price:      7089736

Curr date is 20111103
Last traded price: 1869.96
Fluctuation ratio: -1.48%
Traded volume:     427533
Traded price:      6206253

Curr date is 20111102
Last traded price: 1898.01
Fluctuation ratio: -0.61%
Traded volume:     383783
Traded price:      6639959

Curr date is 20111101
Last traded price: 1909.63
Fluctuation ratio: +0.03%
Traded volume:     348709
Traded price:      6196173

Curr date is 20111031
Last traded price: 1909.03
Fluctuation ratio: -1.06%
Traded volume:     387011
Traded price:      6255819

Curr date is 20111028
Last traded price: 1929.48
Fluctuation ratio: +0.39%
Traded volume:     504187
Traded price:      10022898
```

```
Curr date is 20111027
Last traded price: 1922.04
Fluctuation ratio: +1.46%
Traded volume:     410379
Traded price:      7817092

Curr date is 20111026
Last traded price: 1894.31
Fluctuation ratio: +0.30%
Traded volume:     379027
Traded price:      5489842

Curr date is 20111025
Last traded price: 1888.65
Fluctuation ratio: -0.51%
Traded volume:     348599
Traded price:      5969582

Curr date is 20111024
Last traded price: 1898.32
Fluctuation ratio: +3.26%
Traded volume:     369269
Traded price:      6664124

Curr date is 20111021
Last traded price: 1838.38
Fluctuation ratio: +1.84%
Traded volume:     350266
Traded price:      6355680

Curr date is 20111020
Last traded price: 1805.09
Fluctuation ratio: -2.74%
Traded volume:     341616
Traded price:      6680274

Curr date is 20111019
Last traded price: 1855.92
Fluctuation ratio: +0.93%
Traded volume:     336112
Traded price:      5695649

Curr date is 20111018
Last traded price: 1838.90
Fluctuation ratio: -1.41%
Traded volume:     313747
Traded price:      5659423

Curr date is 20111017
Last traded price: 1865.18
Fluctuation ratio: +1.62%
Traded volume:     363287
Traded price:      5842287
```

```
Curr date is 20111014
Last traded price: 1835.40
Fluctuation ratio: +0.67%
Traded volume:     344596
Traded price:      4990769

Curr date is 20111013
Last traded price: 1823.10
Fluctuation ratio: +0.75%
Traded volume:     425444
Traded price:      6529655

Curr date is 20111012
Last traded price: 1809.50
Fluctuation ratio: +0.81%
Traded volume:     346648
Traded price:      5591456

Curr date is 20111011
Last traded price: 1795.02
Fluctuation ratio: +1.62%
Traded volume:     394217
Traded price:      7030870

Curr date is 20111010
Last traded price: 1766.44
Fluctuation ratio: +0.38%
Traded volume:     271702
Traded price:      4968059

Curr date is 20111007
Last traded price: 1759.77
Fluctuation ratio: +2.89%
Traded volume:     329710
Traded price:      7442564

Curr date is 20111006
Last traded price: 1710.32
Fluctuation ratio: +2.63%
Traded volume:     347674
Traded price:      7770663

Curr date is 20111005
Last traded price: 1666.52
Fluctuation ratio: -2.33%
Traded volume:     330973
Traded price:      7560971

Curr date is 20111004
Last traded price: 1706.19
Fluctuation ratio: -3.59%
Traded volume:     296709
```

```
Traded price:      6599426

Curr date is 20110930
Last traded price: 1769.65
Fluctuation ratio: +0.02%
Traded volume:     322160
Traded price:      6149813

Curr date is 20110929
Last traded price: 1769.29
Fluctuation ratio: +2.68%
Traded volume:     369425
Traded price:      6594996

Curr date is 20110928
Last traded price: 1723.09
Fluctuation ratio: -0.73%
Traded volume:     339547
Traded price:      6248654

Curr date is 20110927
Last traded price: 1735.71
Fluctuation ratio: +5.02%
Traded volume:     334476
Traded price:      6098523

Curr date is 20110926
Last traded price: 1652.71
Fluctuation ratio: -2.64%
Traded volume:     419143
Traded price:      7283338

Curr date is 20110923
Last traded price: 1697.44
Fluctuation ratio: -5.73%
Traded volume:     407148
Traded price:      7442827

Curr date is 20110922
Last traded price: 1800.55
Fluctuation ratio: -2.90%
Traded volume:     291142
Traded price:      5705571

Curr date is 20110921
Last traded price: 1854.28
Fluctuation ratio: +0.89%
Traded volume:     293377
Traded price:      5877761

Curr date is 20110920
Last traded price: 1837.97
Fluctuation ratio: +0.94%
```

```
Traded volume:     317144
Traded price:      5461918

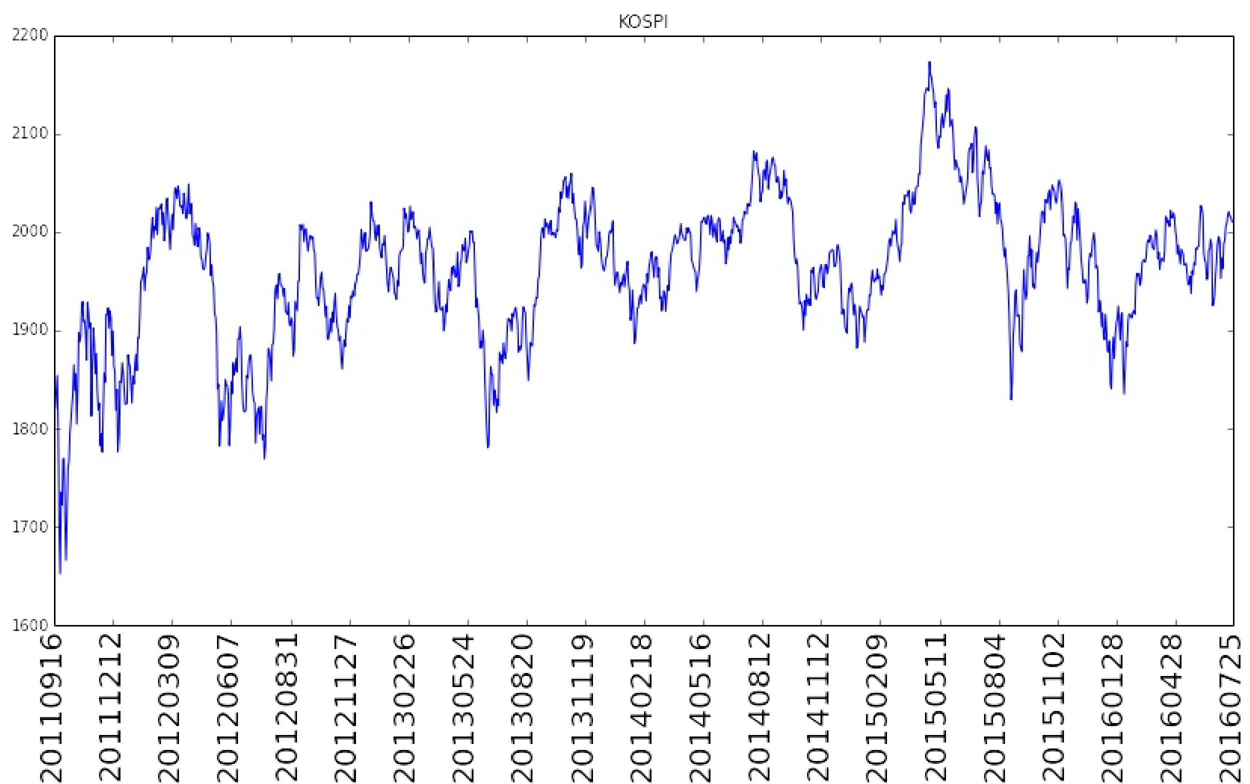
Curr date is 20110919
Last traded price: 1820.94
Fluctuation ratio: -1.04%
Traded volume:     331500
Traded price:      4916366

Curr date is 20110916
Last traded price: 1840.10
Fluctuation ratio: +3.72%
Traded volume:     423993
Traded price:      7510863
```

Plot

```
plt.figure(1, figsize=(14, 7))
n = len(last_trade_prices)
x = range(n)
step = int(n/20)
xtick_input = x[0:n:step]
xtick_input.append(x[-1])
xtick_str = date_strs[0:n:step]
xtick_str.append(date_strs[-1])
plt.xticks(xtick_input, xtick_str, fontsize=20
           , rotation='vertical')
plt.plot(x, last_trade_prices, "-")
plt.title("KOSPI")
```

```
<matplotlib.text.Text at 0x7f02aadc3350>
```



GPR on this data

Kernel function

```
def kernel_se(X1, X2, g2, l2, w2):
    n1 = X1.shape[0]
    n2 = X2.shape[0]
    K  = np.zeros((n1, n2))
    for i in range(n1):
        for j in range(n2):
            x1 = X1[i, :]
            x2 = X2[j, :]
            d  = x1 - x2
            K[i, j] = g2*np.exp(-d*d/(l2))
            if n1 > 1 and i == j:
                K[i, j] = K[i, j] + w2
    return K
print ("Kernel function defined")
```

```
Kernel function defined
```

GPR

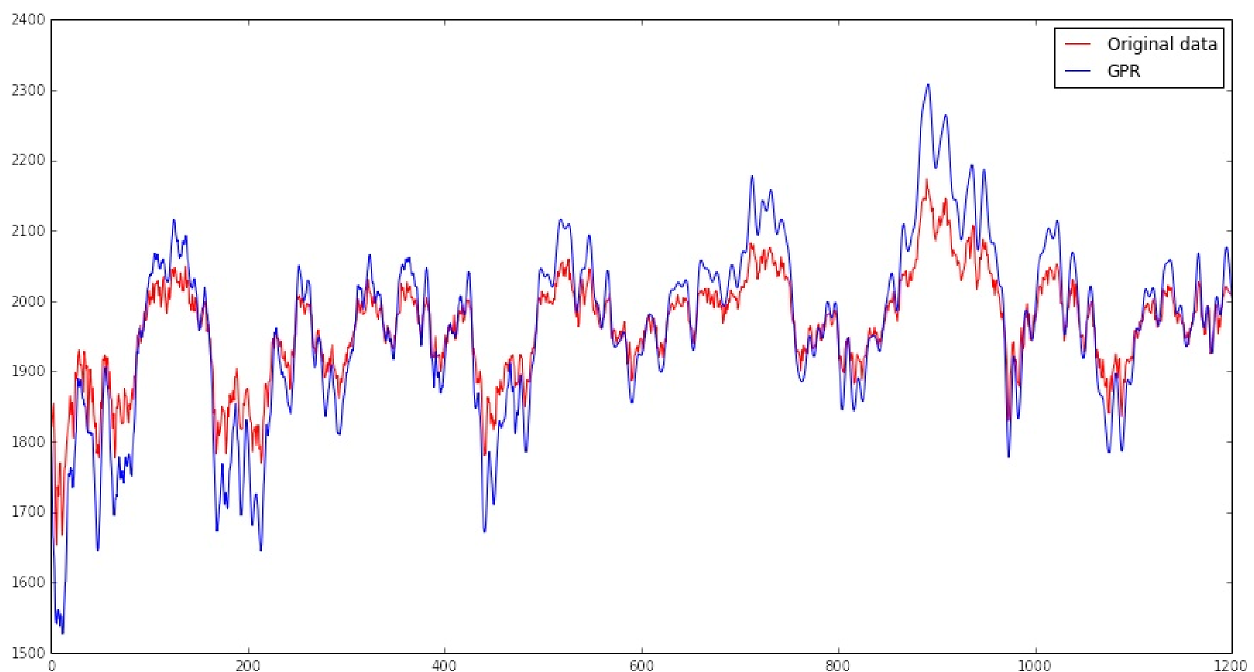
```
xdata = np.array(x).reshape(-1, len(x)).T
ydata = np.array(last_trade_prices).reshape(-1, len(last_trade_p
rices)).T
ydata_mz = ydata - np.mean(ydata)
xtest = xtest = np.array([np.linspace(0, 1200, 3000)]).T
g2    = 1
l2    = 30
w2    = 1e-1
Kdata = kernel_se(xdata, xdata, g2, l2, w2)
alpha = np.matmul(np.linalg.inv(Kdata), ydata_mz)
Ktest = kernel_se(xtest, xdata, g2, l2, w2)
ytest = np.matmul(Ktest, alpha) + np.mean(ydata)
print ("GPR ready")

# Plot
plt.figure(1, figsize=(15, 8))
plt.plot(xdata[:, 0], ydata[:, 0], 'r-', label='Original data')
plt.plot(xtest[:, 0], ytest[:, 0], 'b-', label='GPR')
plt.legend()
```

```
GPR ready



<matplotlib.legend.Legend at 0x7f02a8a7c850>
```




```
import pandas
```

Gaussian process regrssion demo

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
print ("Packages Loaded")
```

```
Packages Loaded
```

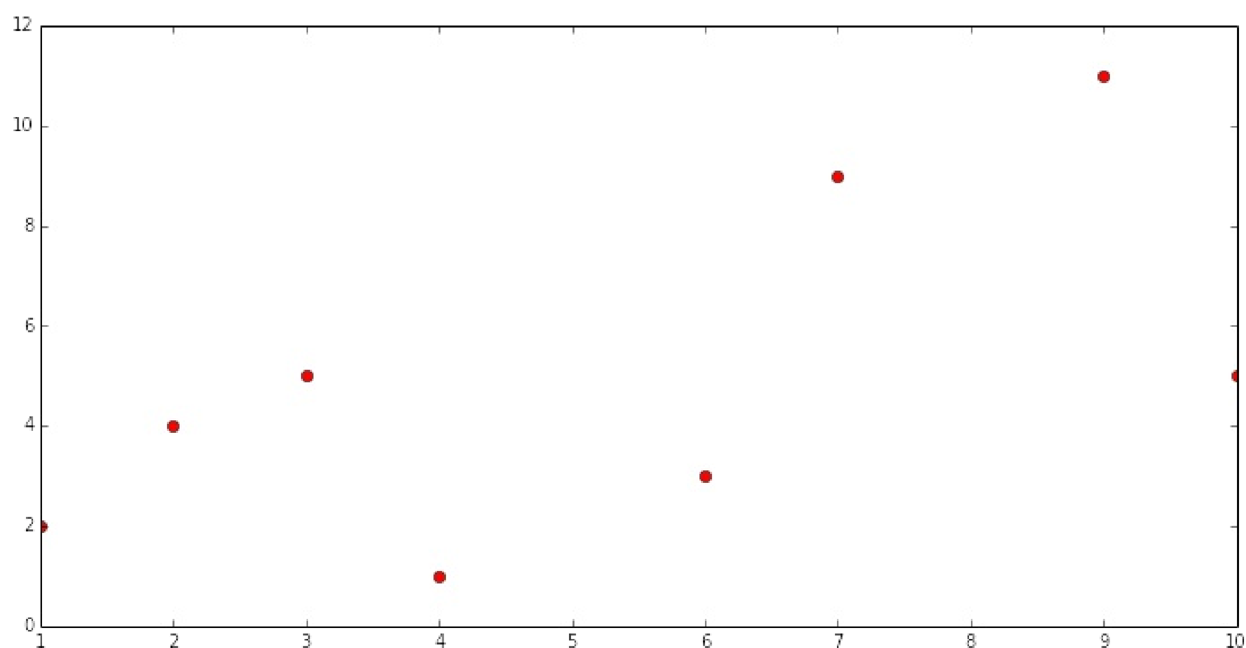
Define training data

```
x = np.array([[1, 2, 3, 4, 6, 7, 9, 10]]).T
y = np.array([[2, 4, 5, 1, 3, 9, 11, 5]]).T

# Plot
plt.figure(1, figsize=(12, 6))
plt.plot(x[:, 0], y[:, 0], 'ro', label='Original data')

print ("Training data")
```

```
Training data
```



Define kernel function

```
def kernel_se(X1, X2):
    n1 = X1.shape[0]
    n2 = X2.shape[0]
    K  = np.zeros((n1, n2))
    for i in range(n1):
        for j in range(n2):
            x1 = X1[i, :]
            x2 = X2[j, :]
            d  = x1 - x2
            K[i, j] = 10*np.exp(-d*d/(10))
            if n1 > 1 and i == j:
                K[i, j] = K[i, j] + 0.001
    return K
print ("Define kernel function")
```

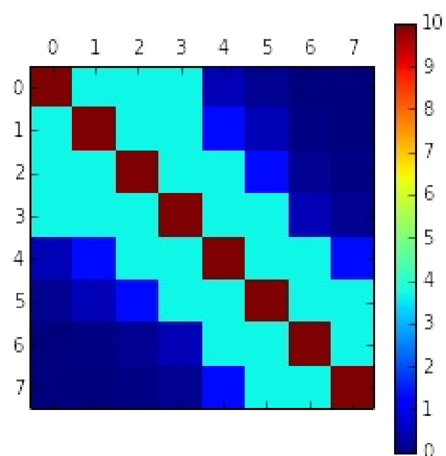
```
Define kernel function
```

Kernel matrix

```
K = kernel_se(x, x)
plt.figure(1, figsize=(6, 6))
plt.matshow(K)
plt.colorbar()
plt.show()
```

```
<matplotlib.figure.Figure at 0x7f7f8d958f50>


/usr/lib/pymodules/python2.7/matplotlib/collections.py:548: Futu
reWarning: elementwise comparison failed; returning scalar inste
ad, but in the future will perform elementwise comparison
  if self._edgecolors == 'face':
```

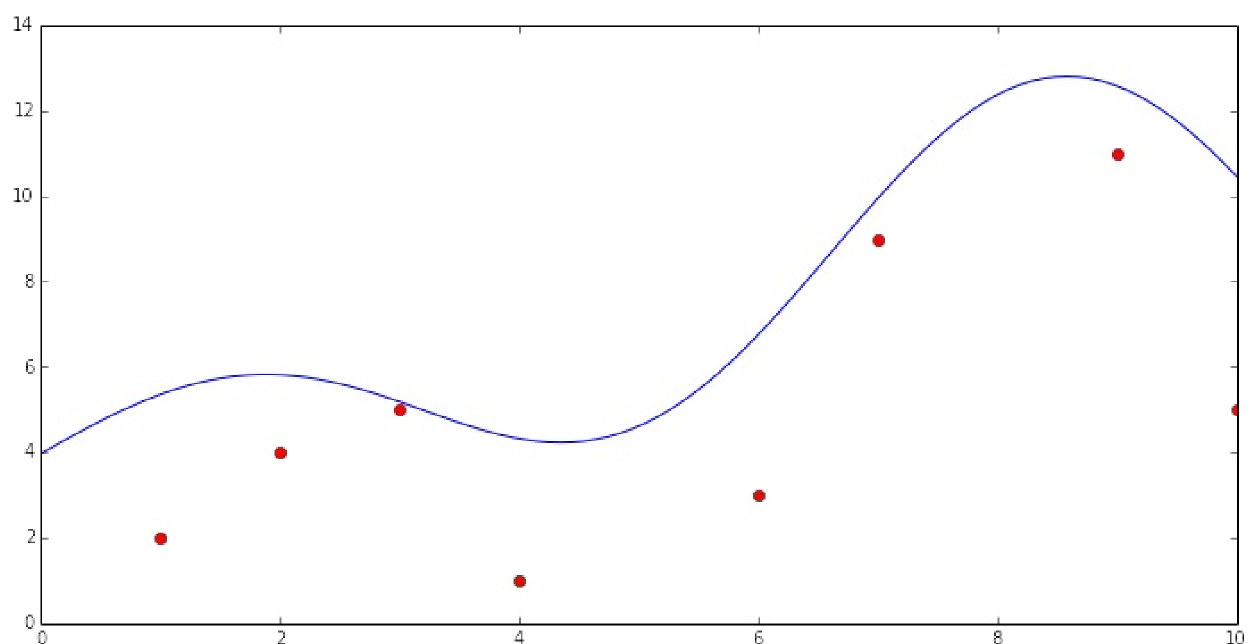


Do Gaussian process regression

```
ntest = 100
xtest = np.array([np.linspace(0, 10, ntest)]).T
Ktest = kernel_se(xtest, x)

# GPR
alpha = np.matmul(np.linalg.inv(K), y)
ytest = np.matmul(Ktest, alpha)
plt.figure(1, figsize=(12, 6))
plt.plot(x[:, 0], y[:, 0], 'ro', label='Original data')
plt.plot(xtest[:, 0], ytest[:, 0], 'b-', label='GPR')
```

```
[<matplotlib.lines.Line2D at 0x7f7f8db4c550>]
```

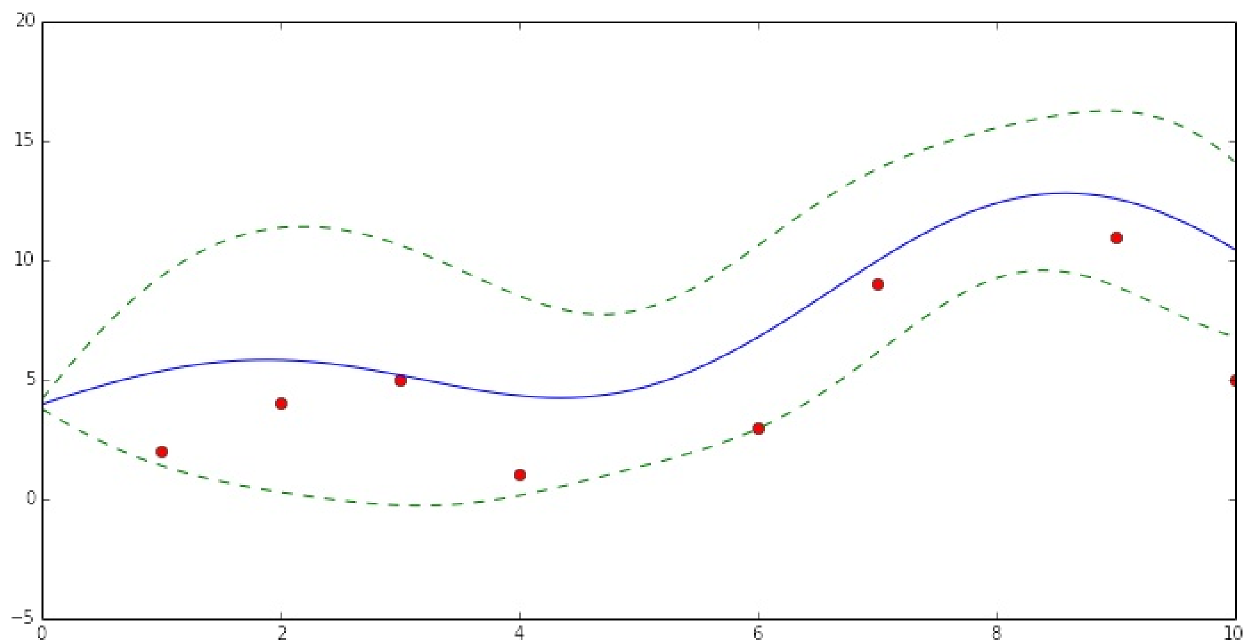


GP Variance

```
vartest = np.zeros((ntest, 1))
for i in range(ntest):
    xtemp  = xtest[i:i+1, :]
    ktemp  = kernel_se(xtemp, xtemp)
    ktemp2 = kernel_se(xtemp, x)
    Mtemp  = np.matmul(ktemp2, np.linalg.inv(K))
    Mtemp2 = np.matmul(Mtemp, ktemp2.T)
    vartest[i, :] = ktemp - Mtemp2

# Plot
plt.figure(1, figsize=(12, 6))
plt.plot(x[:, 0], y[:, 0], 'ro', label='Original data')
plt.plot(xtest[:, 0], ytest[:, 0], 'b-', label='GPR')
plt.plot(xtest[:, 0], ytest[:, 0] + vartest[:, 0], 'g--', label=
'GPR')
plt.plot(xtest[:, 0], ytest[:, 0] - vartest[:, 0], 'g--', label=
'GPR')
```

```
[<matplotlib.lines.Line2D at 0x7f7f8d968a10>]
```



Neural style

This is a simple refactoring of Anish Athalye's neural style

```
import scipy.io
import numpy as np
import os
import scipy.misc
import matplotlib.pyplot as plt
import tensorflow as tf
%matplotlib inline
print ("Packages loaded")
```

```
Packages loaded
```

Define VGG

```
def net(data_path, input_image):
    layers = (
        'conv1_1', 'relu1_1', 'conv1_2', 'relu1_2', 'pool1',
        'conv2_1', 'relu2_1', 'conv2_2', 'relu2_2', 'pool2',
        'conv3_1', 'relu3_1', 'conv3_2', 'relu3_2', 'conv3_3',
        'relu3_3', 'conv3_4', 'relu3_4', 'pool3',
        'conv4_1', 'relu4_1', 'conv4_2', 'relu4_2', 'conv4_3',
        'relu4_3', 'conv4_4', 'relu4_4', 'pool4',
        'conv5_1', 'relu5_1', 'conv5_2', 'relu5_2', 'conv5_3',
        'relu5_3', 'conv5_4', 'relu5_4'
    )
    data = scipy.io.loadmat(data_path)
    mean = data['normalization'][0][0][0]
    mean_pixel = np.mean(mean, axis=(0, 1))
    weights = data['layers'][0]
    net = {}
    current = input_image
    for i, name in enumerate(layers):
        kind = name[:4]
        if kind == 'conv':
            kernels, bias = weights[i][0][0][0][0]
            # matconvnet: weights are [width, height, in_channel
s, out_channels]
            # tensorflow: weights are [height, width, in_channel
s, out_channels]
            kernels = np.transpose(kernels, (1, 0, 2, 3))
```

```
            bias = bias.reshape(-1)
            current = _conv_layer(current, kernels, bias)
        elif kind == 'relu':
            current = tf.nn.relu(current)
        elif kind == 'pool':
            current = _pool_layer(current)
        net[name] = current
    assert len(net) == len(layers)
    return net, mean_pixel, layers
def _conv_layer(input, weights, bias):
    conv = tf.nn.conv2d(input, tf.constant(weights), strides=(1,
1, 1, 1),
            padding='SAME')
    return tf.nn.bias_add(conv, bias)
def _pool_layer(input):
    return tf.nn.max_pool(input, ksize=(1, 2, 2, 1), strides=(1,
2, 2, 1),
            padding='SAME')
def preprocess(image, mean_pixel):
    return image - mean_pixel
def unprocess(image, mean_pixel):
    return image + mean_pixel
def imread(path):
    return scipy.misc.imread(path).astype(np.float)
def imsave(path, img):
    img = np.clip(img, 0, 255).astype(np.uint8)
    scipy.misc.imsave(path, img)
print ("Network for VGG ready")
```

```
Network for VGG ready
```

Extract features of a content image

```
cwd           = os.getcwd()
VGG_PATH      = cwd + "/data/imagenet-vgg-verydeep-19.mat"
CONTENT_PATH  = cwd + "/images/flash.jpg"
CONTENT_LAYER = 'relu2_2'
STYLE_LAYERS  = ('relu1_1', 'relu2_1', 'relu3_1', 'relu4_1', 're
lu5_1')
# STYLE_LAYERS  = ('relu1_1', 'relu2_1')

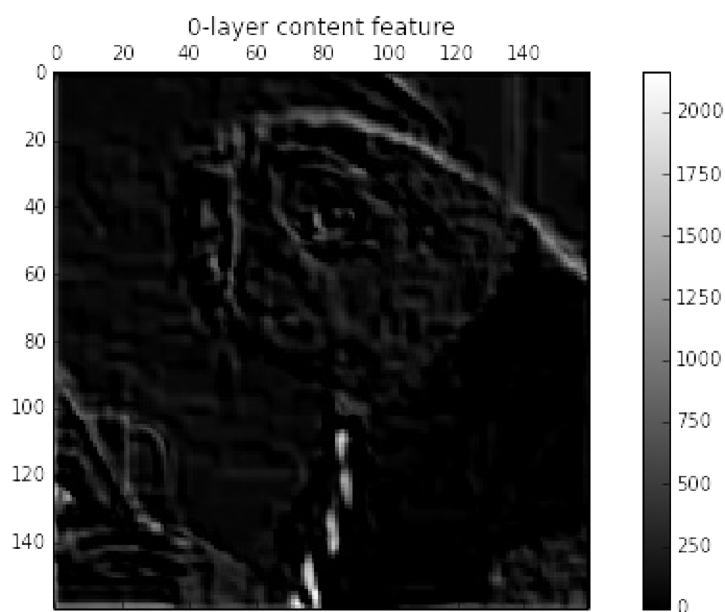
raw_content = scipy.misc.imread(CONTENT_PATH)
plt.figure(0, figsize=(10, 5))
plt.imshow(raw_content)
plt.title("Original content image")
plt.show()

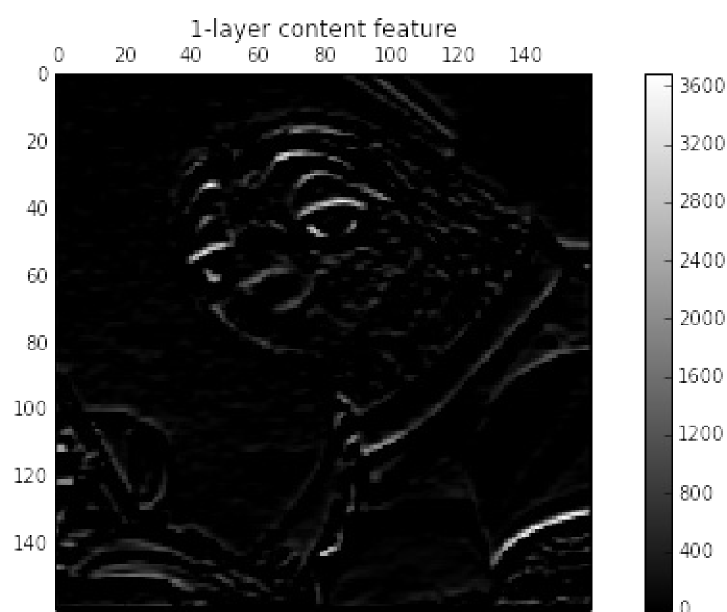
content_image = raw_content.astype(np.float)
content_shape = (1,) + content_image.shape # (h, w, nch) =>  (1,
 h, w, nch)
with tf.Graph().as_default(), tf.Session() as sess, tf.device('/
gpu:0'):
    image = tf.placeholder('float', shape=content_shape)
    nets, content_mean_pixel, _ = net(VGG_PATH, image)
    content_image_pre = np.array([preprocess(content_image, cont
ent_mean_pixel)])
    content_features = nets[CONTENT_LAYER].eval(feed_dict={image
: content_image_pre})
    print (" Type of 'features' is ", type(content_features))
    print (" Shape of 'features' is %s" % (content_features.shap
e,))
    # Plot response
    for i in range(5):
        plt.figure(i, figsize=(10, 5))
        plt.matshow(content_features[0, :, :, i], cmap=plt.cm.gr
ay, fignum=i)
        plt.title("%d-layer content feature" % (i))
        plt.colorbar()
        plt.show()
```

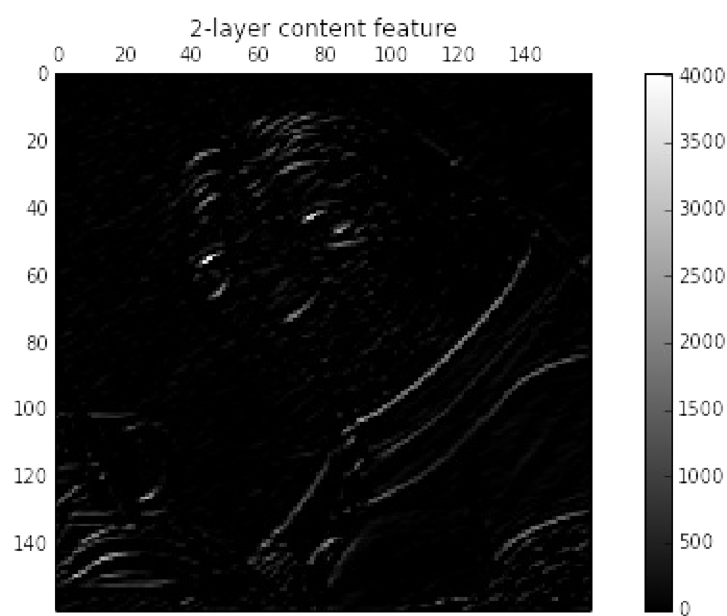


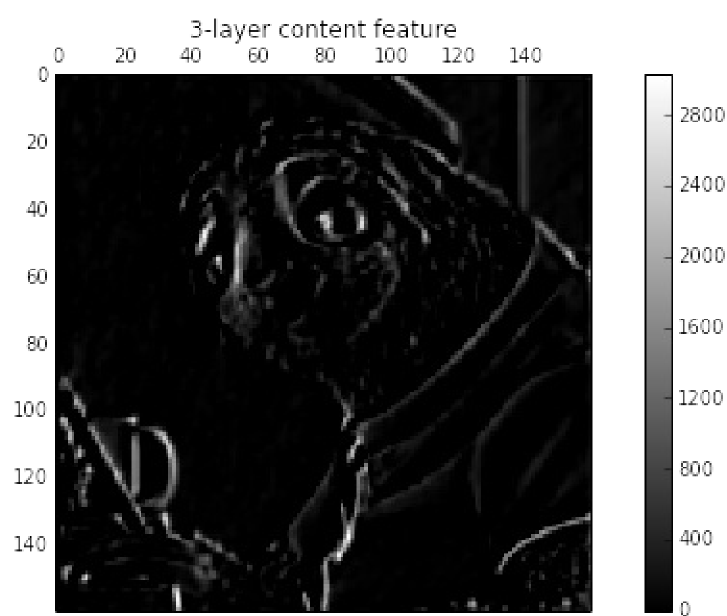

```
(" Type of 'features' is ", <type 'numpy.ndarray'>)
 Shape of 'features' is (1, 160, 160, 128)

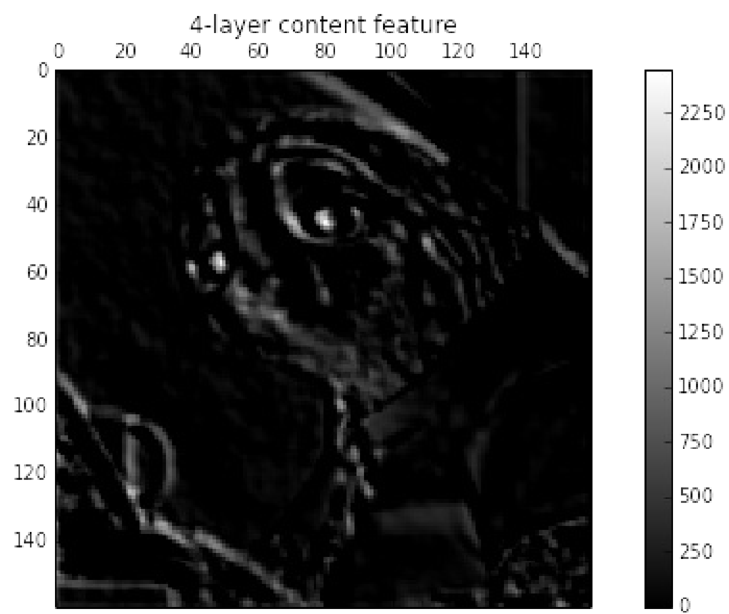

/usr/lib/pymodules/python2.7/matplotlib/collections.py:548: Futu
reWarning: elementwise comparison failed; returning scalar inste
ad, but in the future will perform elementwise comparison
  if self._edgecolors == 'face':
```




1-layer content feature
0
20
40
60
80
100
120
140
3600
3200
2800
2400
2000
1600
1200
800
400
0


2-layer content feature
0
20
40
60
80
100
120
140
4000
3500
3000
2500
2000
1500
1000
500
0

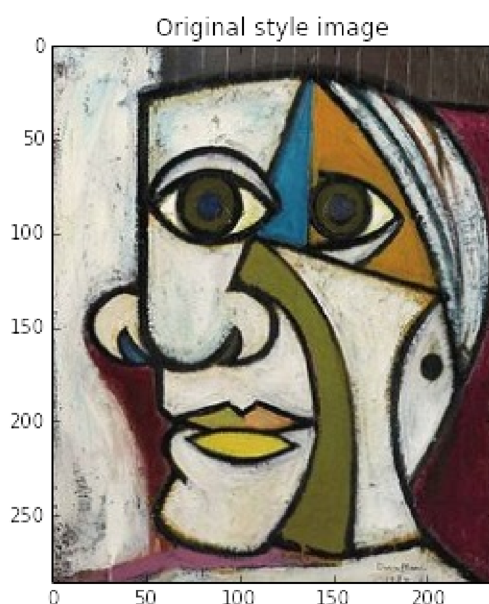

3-layer content feature
0
20
40
60
80
100
120
140
2800
2400
2000
1600
1200
800
400
0

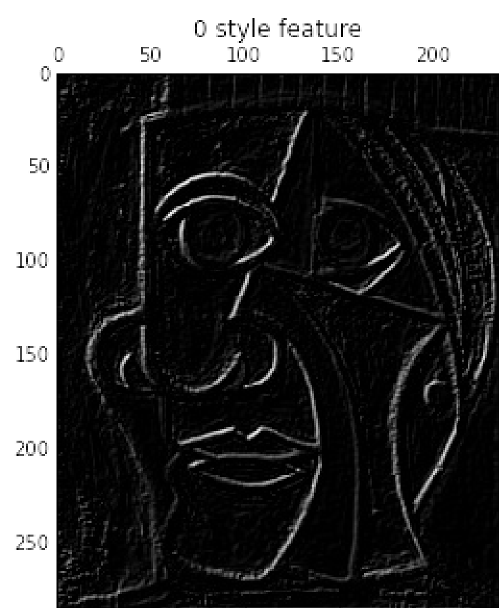


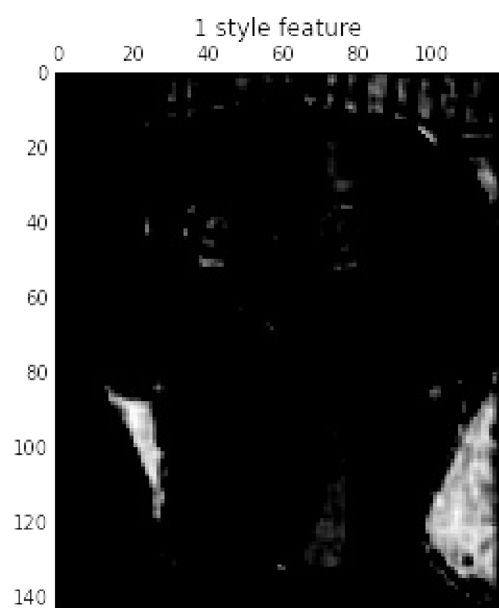
Extract features of a style image

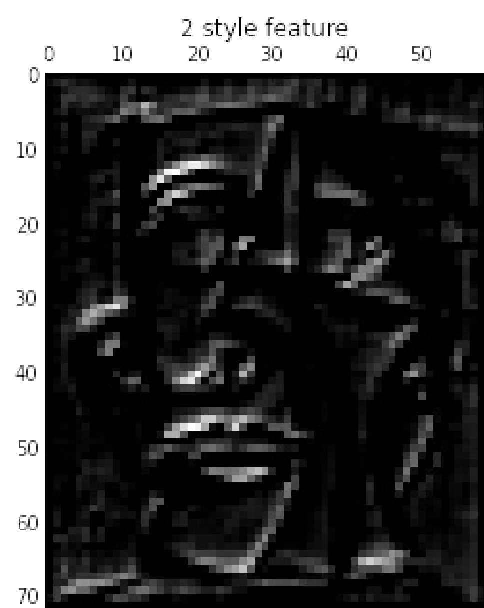
```
STYLE_PATH    = cwd + "/images/style1.jpg"
raw_style     = scipy.misc.imread(STYLE_PATH)
plt.figure(0, figsize=(10, 5))
plt.imshow(raw_style)
plt.title("Original style image")
plt.show()

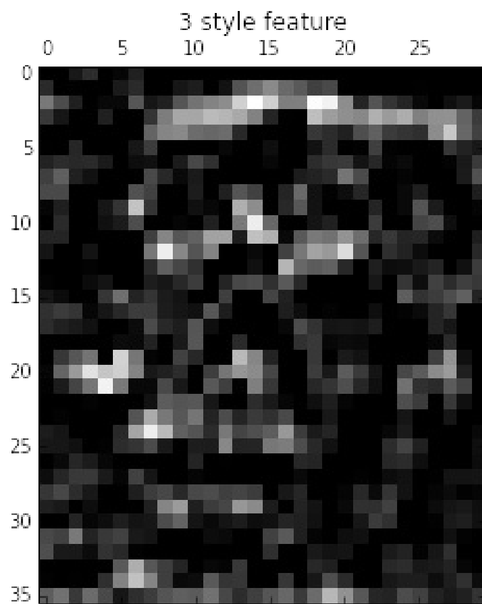
style_image = raw_style.astype(np.float)
style_shape = (1,) + style_image.shape # (h, w, nch) =>  (1, h,
w, nch)
style_features = dict()
with tf.Graph().as_default(), tf.Session() as sess, tf.device('/
gpu:0'):
    image = tf.placeholder('float', shape=style_shape)
    nets, _, _ = net(VGG_PATH, image)
    style_image_pre = np.array([preprocess(style_image, content_
mean_pixel)])
    for idx, layer in enumerate(STYLE_LAYERS):
        curr_features = nets[layer].eval(feed_dict={image: style
_image_pre})
        curr_features_vec = np.reshape(curr_features, (-1, curr_
features.shape[3]))
        gram = np.matmul(curr_features_vec.T, curr_features_vec)
 / curr_features_vec.size
        style_features.update({layer: gram})
        # Plot
        plt.figure(idx, figsize=(10, 5))
        plt.matshow(curr_features[0, :, :, 0], cmap=plt.cm.gray,
 fignum=idx)
        plt.title("%d style feature" % (idx))
        plt.show()
```

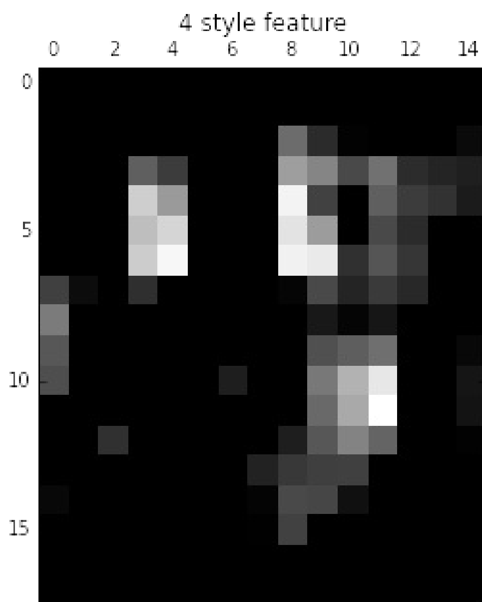



0 style feature
0
50
100
150
200
250


1 style feature
0
20
40
60
80
100
120
140


2 style feature
0
10
20
30
40
50
60
70





Optimize

```
content_weight = 5 # 5
style_weight   = 10 # 10
tv_weight      = 100 # 100
learning_rate  = 5.
iterations     = 1000
def _tensor_size(tensor):
    from operator import mul
    return reduce(mul, (d.value for d in tensor.get_shape()), 1)

with tf.Graph().as_default(), tf.Session() as sess, tf.device('/
gpu:0'):
    initial   = tf.random_normal(content_shape) * 0.256
```

```
    image2opt = tf.Variable(initial)
    nets, mean_pixel, _ = net(VGG_PATH, image2opt)

    # 1. content loss
    content_loss = content_weight * (2 * tf.nn.l2_loss(
        nets[CONTENT_LAYER] - content_features) / content_featur
es.size)

    # 2. style loss
    style_losses = []
    for style_layer in STYLE_LAYERS:
        layer   = nets[style_layer]
        _, height, width, number = layer.get_shape()
        size    = height * width * number
        feats   = tf.reshape(layer, (-1, number.value))
        gram    = tf.matmul(tf.transpose(feats), feats) / size.v
alue
        style_gram = style_features[style_layer]
        style_losses.append(2 * tf.nn.l2_loss(gram - style_gram)
 / style_gram.size)
    style_loss = style_weight * reduce(tf.add, style_losses)

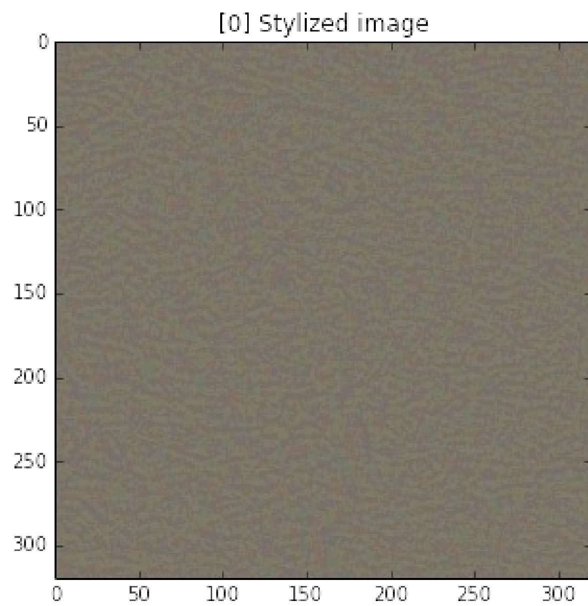
    # 3. Total variation denoising
    tv_y_size = _tensor_size(image2opt[:,1:,:,:])
    tv_x_size = _tensor_size(image2opt[:,:,1:,:])
    tv_loss = tv_weight * 2 * (
        (tf.nn.l2_loss(image2opt[:,1:,:,:] - image2opt[:,:conten
t_shape[1]-1,:,:]) /
            tv_y_size) +
        (tf.nn.l2_loss(image2opt[:,:,1:,:] - image2opt[:,:,:cont
ent_shape[2]-1,:]) /
            tv_x_size))

    # Overall loss
    loss = content_loss + style_loss + tv_loss

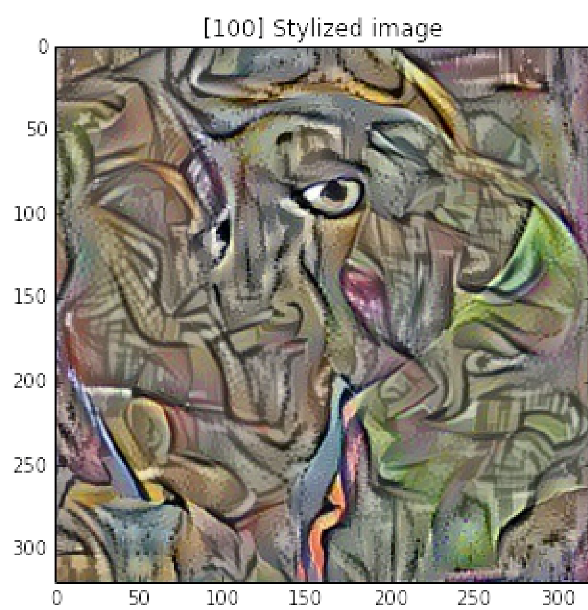
    # optimizer setup
    optm = tf.train.AdamOptimizer(learning_rate).minimize(loss)
    sess.run(tf.initialize_all_variables())
    for i in range(iterations):
        optm.run()
        if i % 100 == 0 or i == iterations-1:
            print ("[%d/%d]" % (i, iterations))
            out = image2opt.eval()
            # Plot
            stylized_img = out[0, :, :, :] + content_mean_pixel
            stylized_img = np.clip(stylized_img, 0, 255).astype(
'uint8')
            plt.figure(0, figsize=(10, 5))
            plt.imshow(stylized_img)
            plt.title("[%d] Stylized image" % (i))
            plt.show()
    out = image2opt.eval()
```

```
print ("done")
```

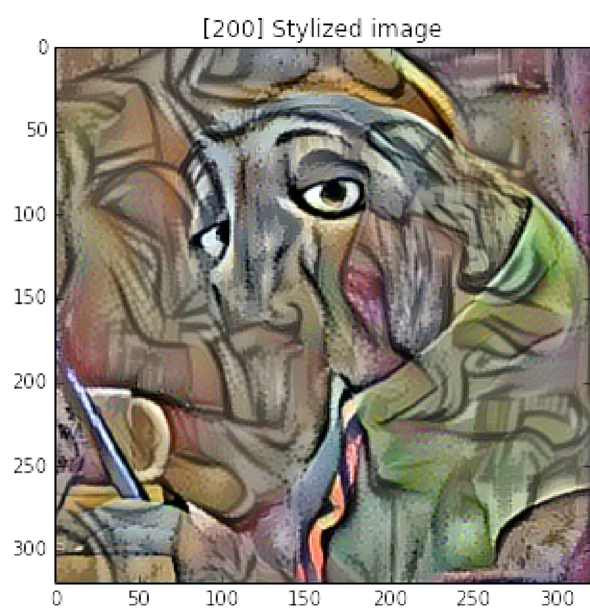
```
[0/1000]
```



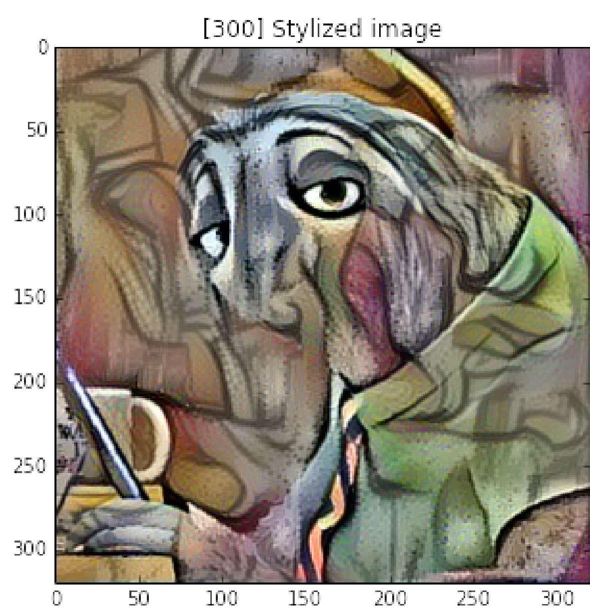

```
[100/1000]
```




```
[200/1000]
```




```
[300/1000]
```




```
[400/1000]
```




```
[500/1000]
```




```
[600/1000]
```




```
[700/1000]
```




```
[800/1000]
```




```
[900/1000]
```



Plot final image

```
plt.figure(0, figsize=(10, 5))
plt.imshow(raw_content)
plt.title("Original content image")
plt.show()

plt.figure(0, figsize=(10, 5))
plt.imshow(raw_style)
plt.title("Original style image")
plt.show()

stylized_img = out[0, :, :, :] + content_mean_pixel
stylized_img = np.clip(stylized_img, 0, 255).astype('uint8')
plt.figure(1, figsize=(10, 5))
plt.imshow(stylized_img)
plt.title("Stylized image")
plt.show()
```

IMPORT PACKAGES

```
import numpy as np
import os
import cv2
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
%matplotlib inline
print ("Packages loaded.")
```

```
Packages loaded.
```

FACE DETECTOR

https://raw.githubusercontent.com/shantnu/Webcam-Face-Detect/master/haarcascade_frontalface_default.xml

```
# Load Face Detector
cwd = os.getcwd()
clsf_path = cwd + "/data/haarcascade_frontalface_default.xml"
face_cascade = cv2.CascadeClassifier(clsf_path)
print ("face_cascade is %s" % (face_cascade))
```

```
face_cascade is <CascadeClassifier 0x7fe089352090>
```

LOAD IMAGE WITH FACES

```
# THIS IS BGR
imgpath = cwd + "/images/celebs.jpg"
# imgpath = cwd + "/../../img_dataset/celebs/Arnold_Schwarzenegg
er/Arnold_Schwarzenegger_0001.jpg"
img_bgr = cv2.imread(imgpath)

# CONVERT TO RGB
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)
```

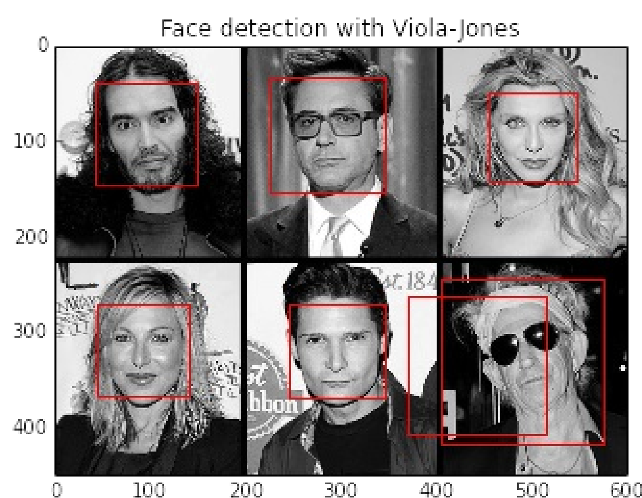
DETECT FACE

```
faces = face_cascade.detectMultiScale(img_gray)
print ("%d faces deteced. " % (len(faces)))
```

```
7 faces deteced.
```

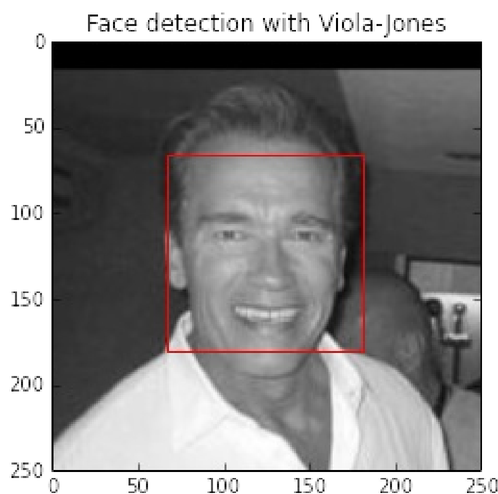
PLOT DETECTED FACES

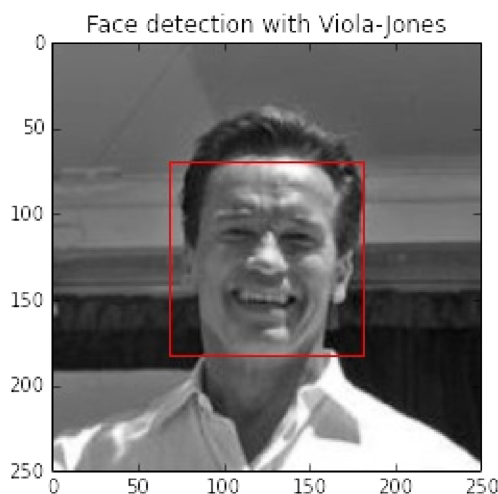
```
# PLOT
plt.figure(0)
plt.imshow(img_gray, cmap=plt.get_cmap("gray"))
ca = plt.gca()
for face in faces:
    ca.add_patch(Rectangle((face[0], face[1]), face[2], face[3]
                                , fill=None, alpha=1, edgecolor='red'
))
plt.title("Face detection with Viola-Jones")
plt.draw()
```

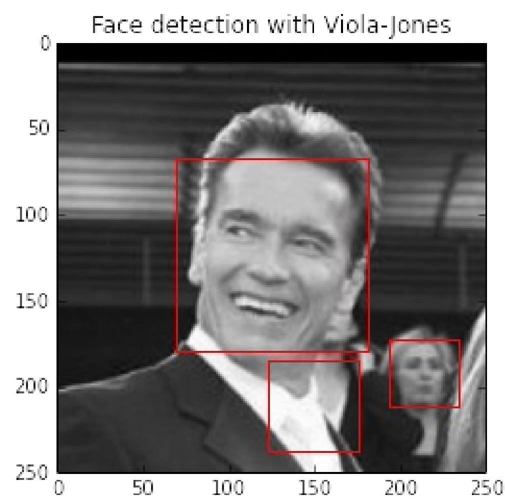


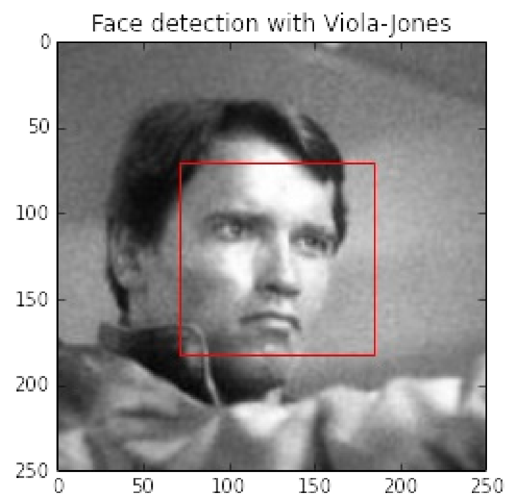
DETECT FACES IN THE IMAGES IN A FOLDER

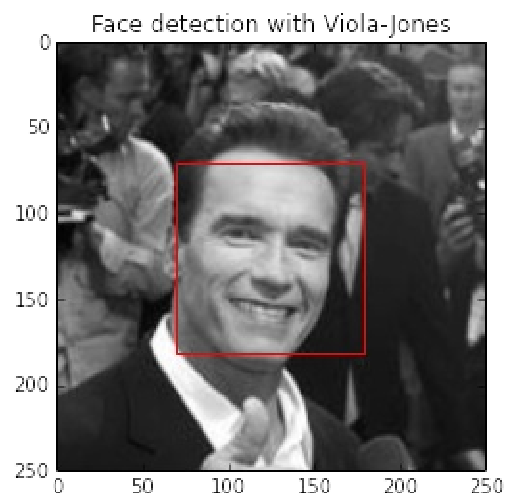
```
path = cwd + "/../../img_dataset/celebs/Arnold_Schwarzenegger"
flist = os.listdir(path)
valid_exts = [".jpg",".gif",".png",".tga", ".jpeg"]
for f in flist:
    if os.path.splitext(f)[1].lower() not in valid_exts:
        continue
    fullpath = os.path.join(path, f)
    img_bgr = cv2.imread(fullpath)
    img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
    img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
    faces = face_cascade.detectMultiScale(img_gray)
    # PLOT
    plt.imshow(img_gray, cmap=plt.get_cmap("gray"))
    ca = plt.gca()
    for face in faces:
        ca.add_patch(Rectangle((face[0], face[1]), face[2], face[
3]
                               , fill=None, alpha=1, edgecolor='
red'))
    plt.title("Face detection with Viola-Jones")
    plt.show()
```

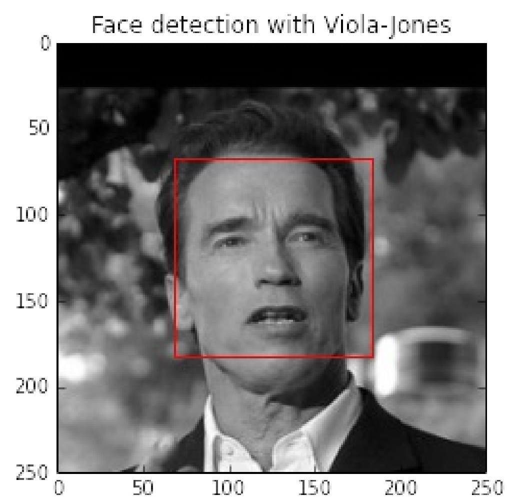


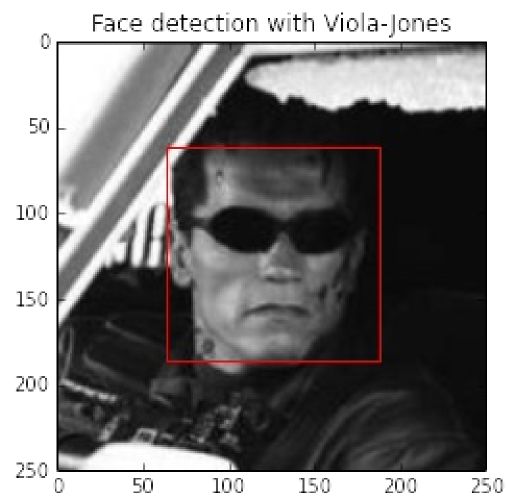


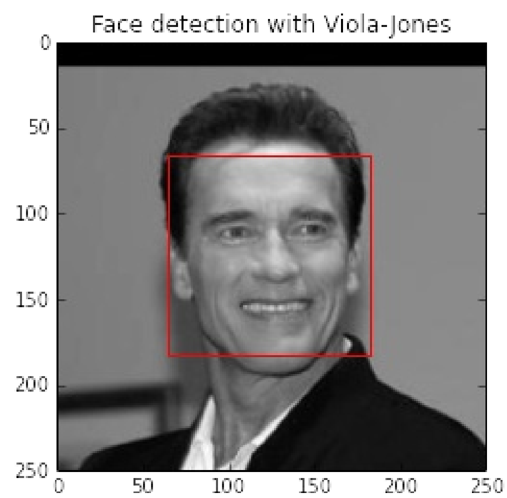

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

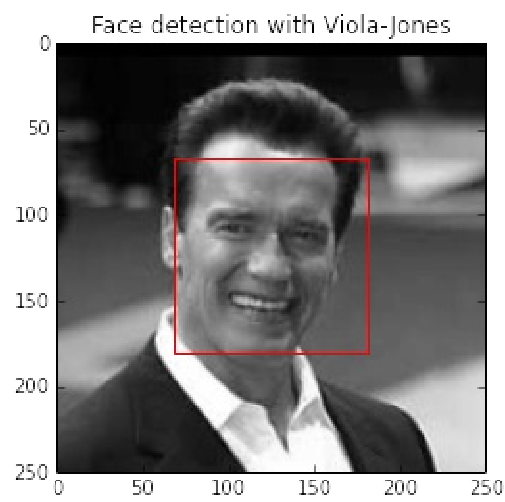

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

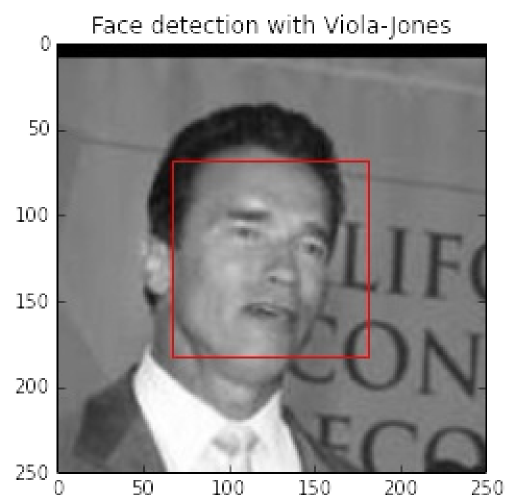

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

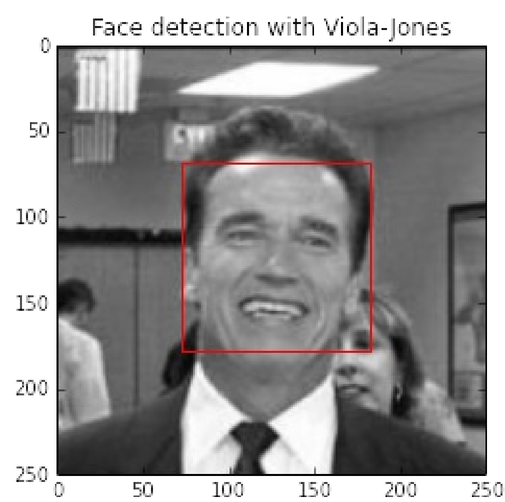

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

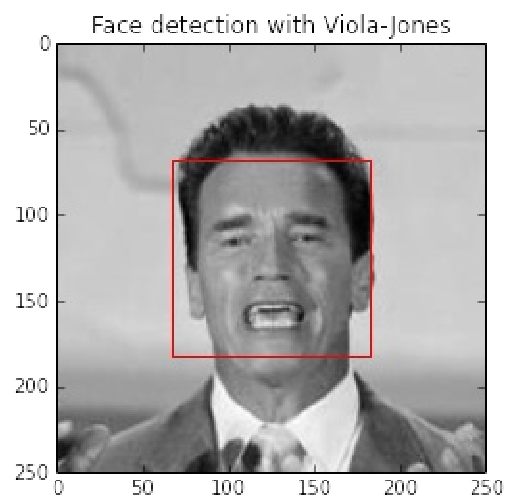

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

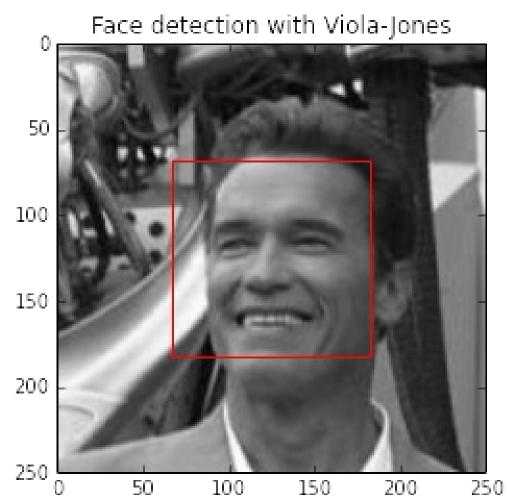

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

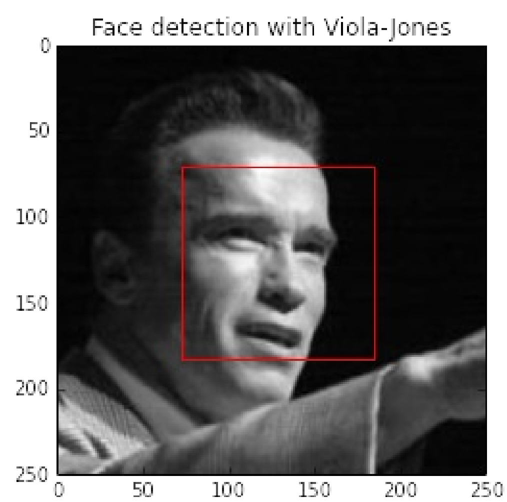

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

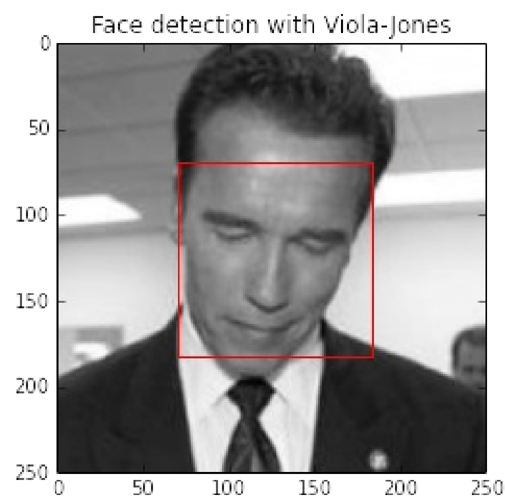

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

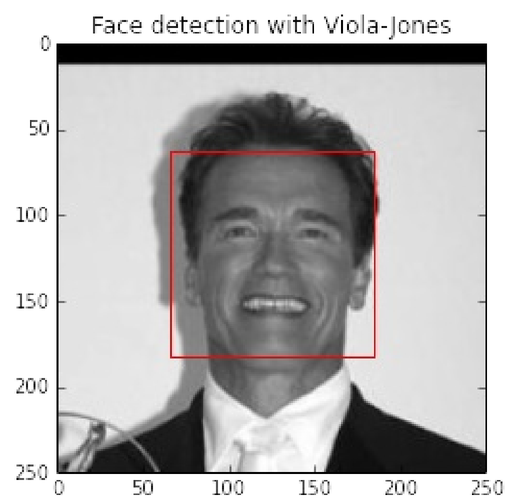

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

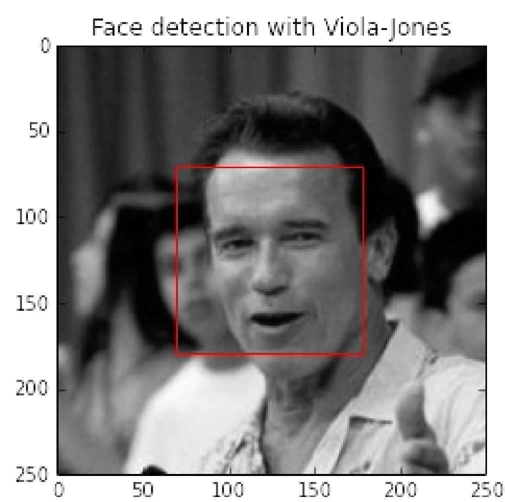

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

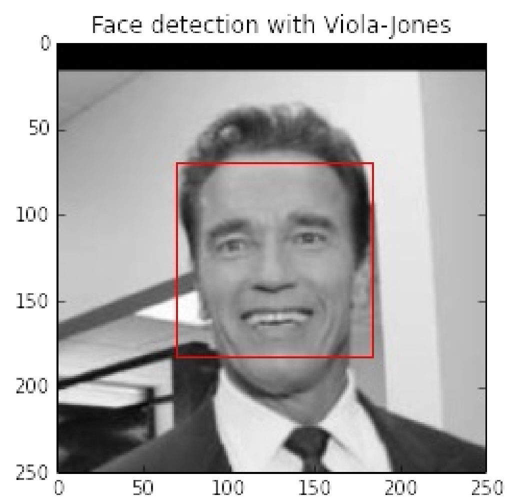

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

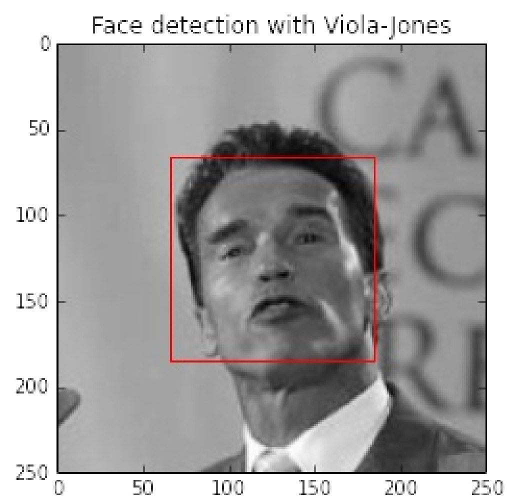

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

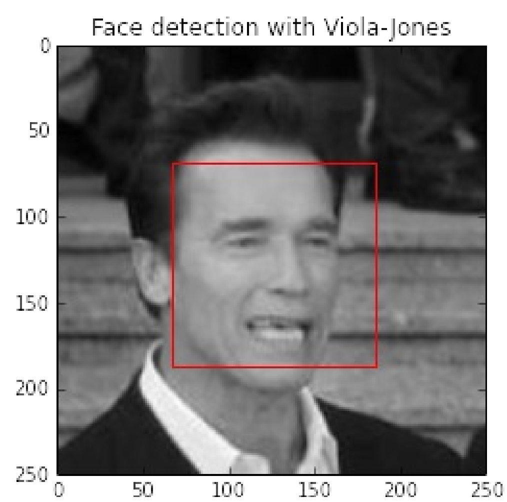

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

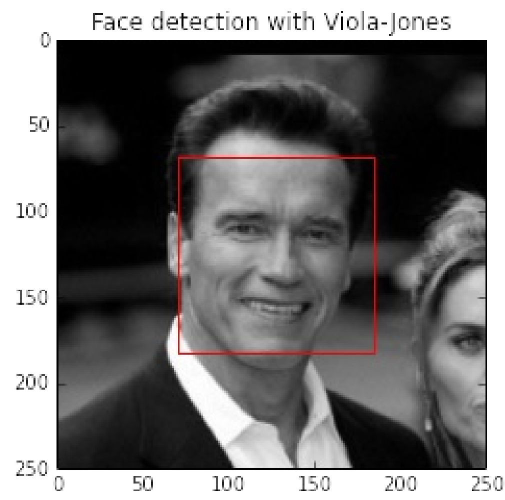

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

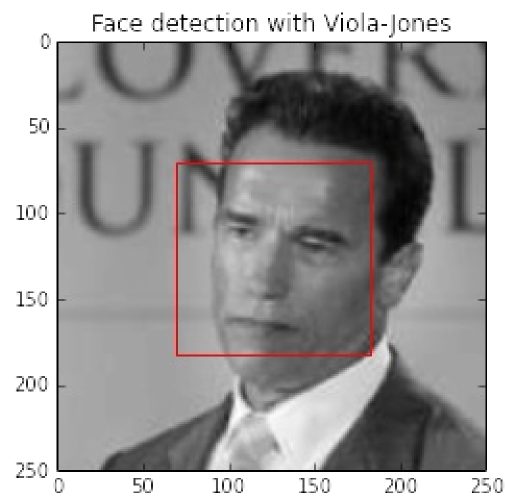

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

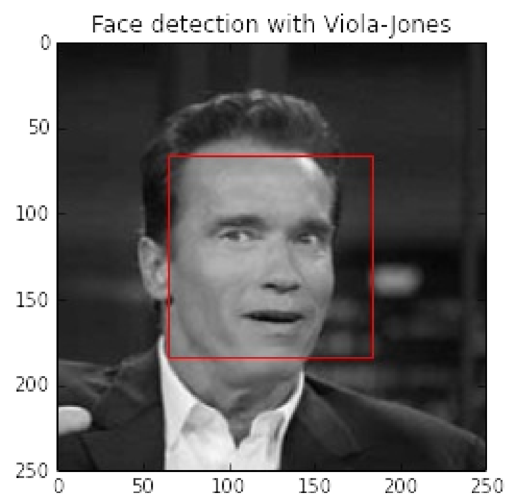

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

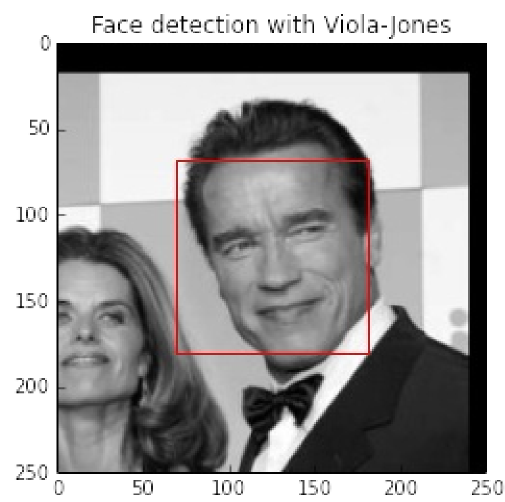

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

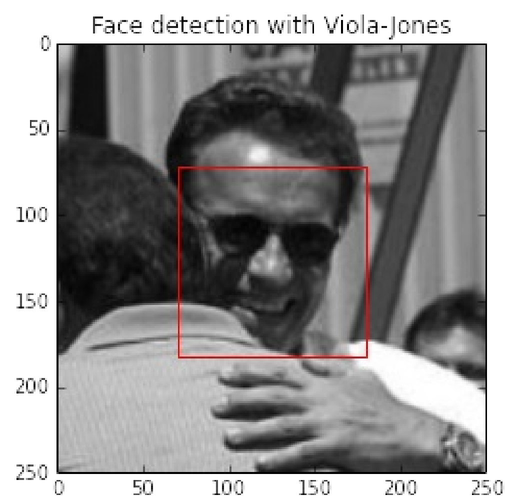

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

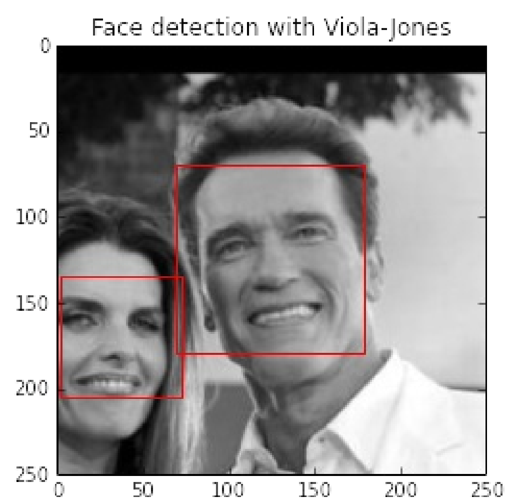

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

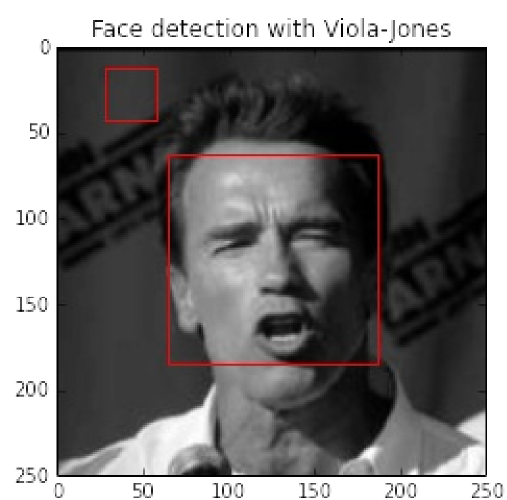

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

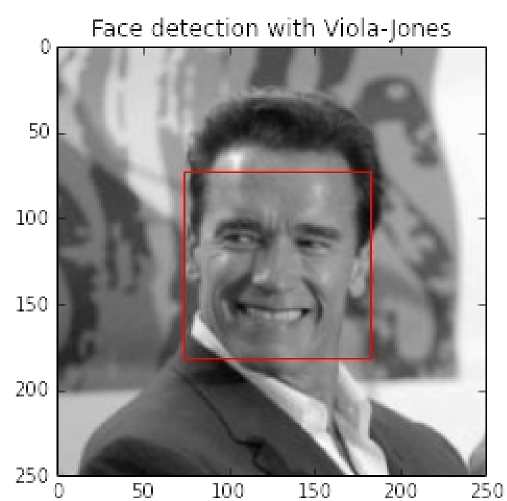

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

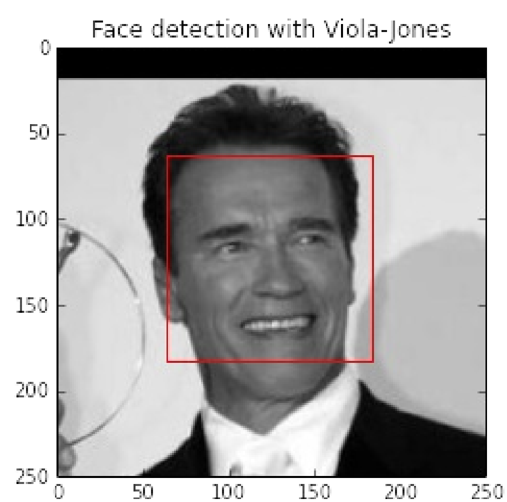

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

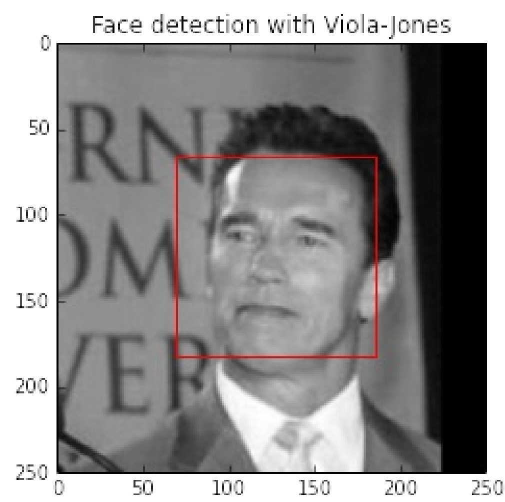

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

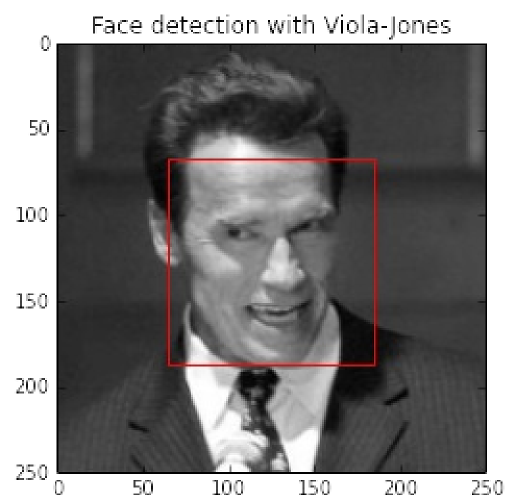

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

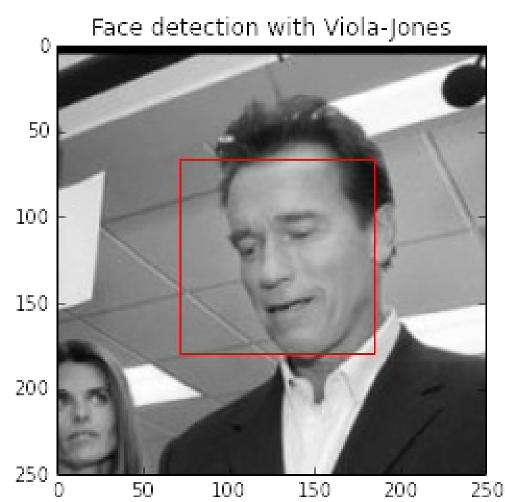

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

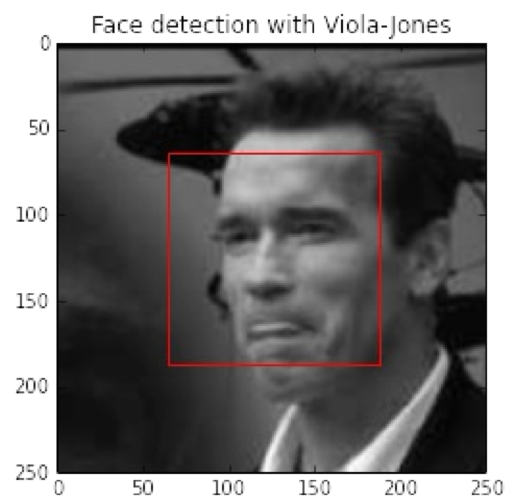

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

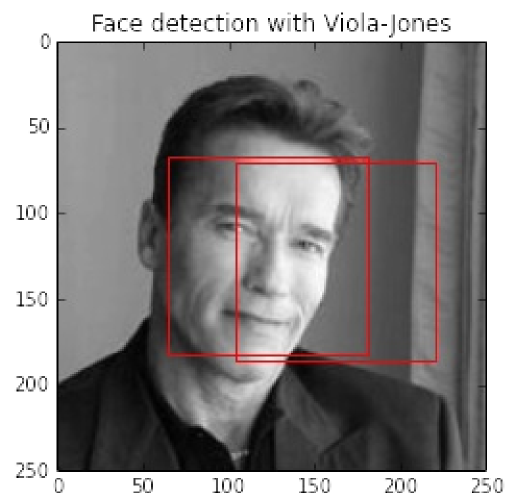

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

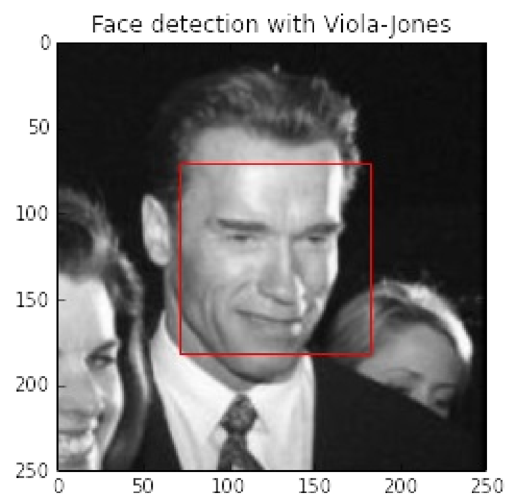

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

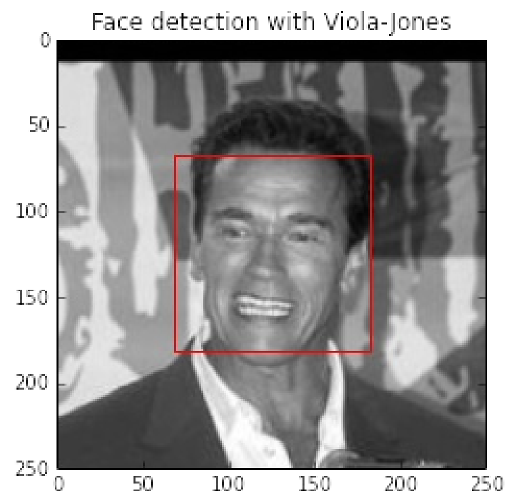

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

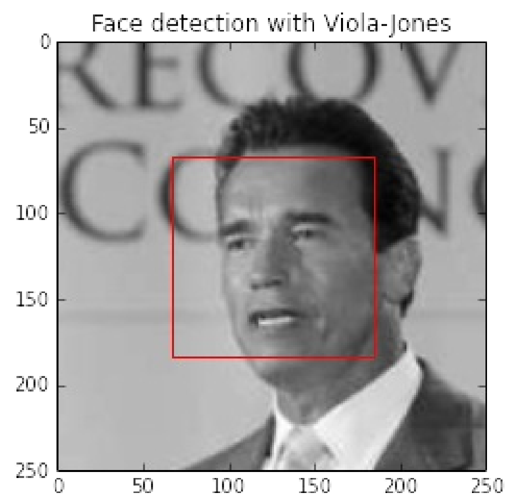

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

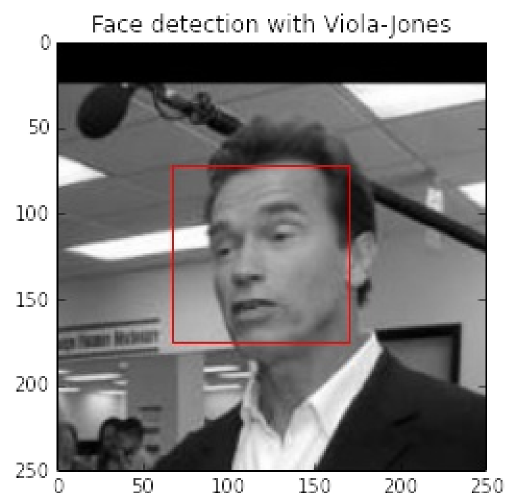

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

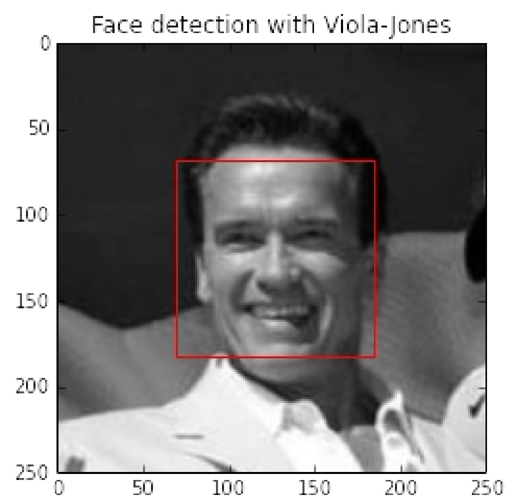

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

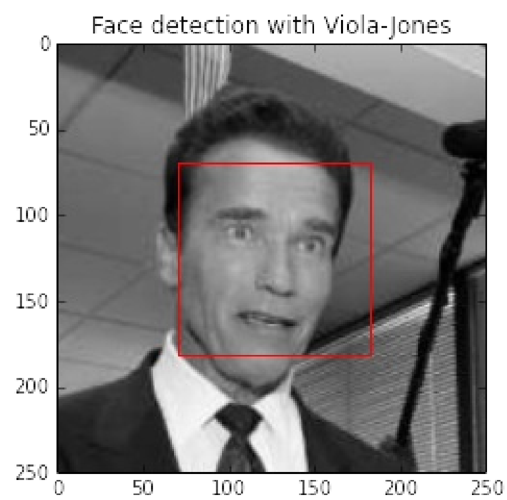

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

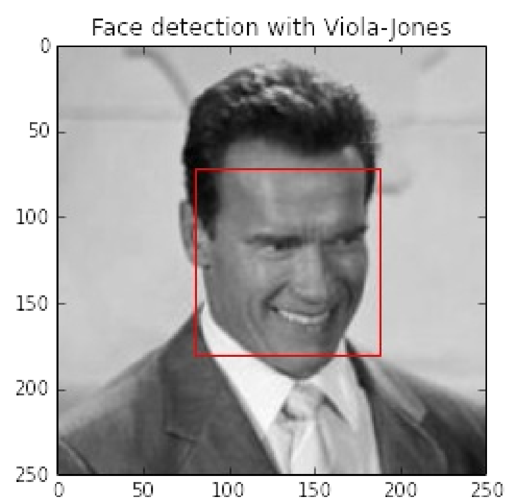

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250

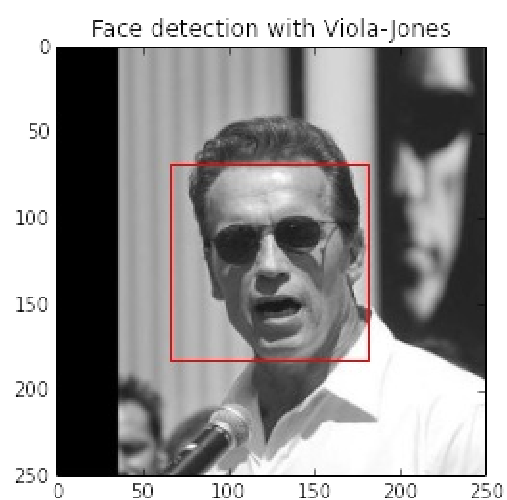

Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250


Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250


Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250


Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250


Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250


Face detection with Viola-Jones
0
50
100
150
200
250
0
50
100
150
200
250